SINGLE-PASS SERIAL SCHEDULING HEURISTIC FOR EGLIN AFB RANGE SERVICES DIVISION SCHEDULE

GRADUATE RESEARCH PROJECT

Matthew Liljenstolpe, Major, USAF

AFIT-IOA-ENS-09C-02

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-IOA-ENS-09C-02

SINGLE-PASS SERIAL SCHEDULING HEURISTIC FOR EGLIN AFB RANGE SERVICES DIVISION SCHEDULE

GRADUATE RESEARCH PROJECT

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Analysis

Matthew Liljenstolpe, BS

Major, USAF

June 2009

AFIT-IOA-ENS-09C-02

SINGLE-PASS SERIAL SCHEDULING HEURISTIC FOR EGLIN AFB RANGE SERVICES DIVISION SCHEDULE

Matthew Liljenstolpe, BS
Major, USAF

Approved:

//signed//	12 Jan 2009
James T. Moore, PhD	date

Abstract

The Air Armament Center (AAC) located at Eglin Air Force Base (AFB) Florida, conducts test and evaluation of United States Air Force (USAF) weapons systems. To enable this, the AAC operates the Eglin Test and Training Complex (ETTC), the largest test range in the United States. InDyne Corporation's Range Services Division (RSD) builds and maintains the infrastructure necessary to conduct world class test and training on the ETTC.

The purpose of this research is to create a scheduling tool for the RSD that maximizes the number of prioritized jobs scheduled and reduces the man-hours required to create a weekly schedule without exceeding a job's deadline, manpower, or equipment constraints. RSD's schedule belongs to a class of scheduling problems called Resource Constrained Project Scheduling Problems (RCPSP). RCPSPs attempt to schedule activities of either a known (deterministic) or variable (stochastic) duration in a defined sequence given a finite amount of resources. Many analytical methods have been created to solve these types of scheduling problems. Analytical solution methods which guarantee optimal solutions were not feasible due to the computational complexity of this RCPSP. Instead, a greedy solution method is explored that uses a single-pass serial scheduling algorithm.

A schedule construction algorithm is provided in the form of pseudo code to enable further research and development of a scheduling tool for this RCPSP. Research on a schedule improvement metaheuristic and coding of the complete algorithm is required before it can be assimilated into existing scheduling software.

To my loving and supportive wife and my very patient children.

Acknowledgments

I'd like to extend my gratitude to the following individuals for their contributions to this document and for the support they have given me during my time at AFIT. For my wife who supports and inspires me through her love and through the drive and determination she tackles every day with. To my children, who exhibit a patience beyond their years. To my fellow students who both pushed me and at time pulled me through some of the more difficult times at AFIT. To Dr. Chambal for his enthusiasm and assistance in supporting our professional goals. To Dr. Moore for his guidance in the completion of this research project. To all the individuals in the 46th Test Wing at Eglin AFB who assisted my research. Special thanks to Lt Col Blatt, Mr. Heald, Mr. Burns, Mr. Wesolowski, Mr. Hutto, and Mrs. Wagner.

Matt Liljenstolpe

Table of Contents

Page

List of Figures

Page

List of Tables

Page

SINGLE-PASS SERIAL SCHEDULING HEURISTIC FOR EGLIN AFB RANGE SERVICES DIVISION SCHEDULE

Chapter 1. Introduction

1.1. Background

Headquarters Air Force Materiel Command (AFMC), at Wright-Patterson AFB Ohio, is responsible for equipping the United States Air Forces (USAF) with warfighting systems to dominate in the Air and Space domains. AFMC provides the personnel and resources necessary to develop and sustain USAF weapon systems throughout their life cycle. An integral aspect of AFMC's mission is to conduct test and evaluation of USAF weapons systems. The Air Armament Center (AAC) located at Eglin AFB Florida is AFMC's preeminent test site for that mission.

The AAC is tasked with the development, acquisition, testing, and sustainment of all air-deliverable weapons in the USAF inventory (Air, 2009). To enable this, the AAC operates the Eglin Test and Training Complex (ETTC), the largest test range in the United States consisting of 724 square miles of land area, over 134,000 square miles of airspace, and more than 123,000 square miles of water ranges in the Gulf of Mexico (Air). Ultimately, the 46th Test Wing (TW) is the AAC unit that provides the expertise and infrastructure necessary to conduct world class test and training on the ETTC.

Like many Department of Defense (DoD) organizations, the 46th Test Wing contracts out certain functions of their mission. InDyne Inc. currently holds the contract to provide management of the ETTC ranges. InDyne's mission is to support ongoing test and training on the ETTC. Every type of ETTC user has unique requirements which

continually stress the resources of the 46th TW. Ultimately, InDyne's Range Support Division (RSD) supplies the majority of the manpower and equipment required to support user requirements. In fact, RSD is the most frequently tasked division of InDyne (Heald, 2009). Specifically, RSD is responsible for all general construction projects plus the maintenance of dozens of facilities on the ETTC ranges.

InDyne's RSD consists of six teams which employ over thirty workers with varying specialties. The teams are organized by specialty and are named: Electricians (E), Welders (W), Carpenters (C), Surveyors (S), Heavy Equipment Operators (H), and Building Maintenance (B). The RSD has an overall division supervisor who manages the master schedule of RSD resources, and each team has a designated team leader who manages the team's jobs and resources. Each year RSD accomplishes over six hundred jobs on the ETTC (Burns, 2009). Scheduling RSD's resources to complete those jobs is a labor intensive and time consuming process. A multitude of dissimilar tasks, frequently changing priorities, shifting work windows, and a limited supply of range time, manpower, and mission critical equipment adds to the complexity of RSD's scheduling processes.

RSD's primary mission is to perform general construction and repair of range assets on the ETTC. To that end, the RSD performs three services for the ETTC, and in order of priority they are: Direct Mission Support (DMS), Range Construction (RC), and Building Maintenance (BM). Jobs classified as DMS involve actions which fulfill requirements for a specific user mission or set of related missions. DMS work usually takes place immediately before or during actual execution of the mission(s). Examples of

DMS include placement and hardening of sensors or cameras, and assembly/disassembly of targets that directly contribute to a mission's execution.

RC projects include but are not limited to target construction, target repair, and test stand construction. The construction and repair of range targets is a continual process and is not usually tied to a particular mission. Although unique targets are sometimes constructed for specific missions, the target and thus those jobs are classified as RC unless they are otherwise tasked as DMS.

BM includes regular maintenance and repair of range facilities controlled and operated by InDyne. BM jobs remain chronically unscheduled because they are continually trumped by higher priority jobs. Subsequently, InDyne is experiencing an increase in the failure of compliance inspections which negatively affect their award bonuses. In order to maintain profitability and provide a high level of service in the long-term, InDyne and consequently the RSD must increase its completion of BM jobs.

1.2. Scope

The scope of this research is limited to the RSD scheduling process. This research focuses on the scheduling of Manpower and Equipment within InDyne's RSD via Task Orders (TOs) and Internal Work Orders (IWOs) which are tasked outside of the 46th TW's DMS schedule.

1.3. Objectives

The objectives of this research are to improve the quality of the RSD schedule as well as decrease the time required to create the initial schedule. Improving the schedule's quality entails maximizing the number of prioritized tasks scheduled within a given time period without exceeding the job's need date (deadline), range time, manpower, or

equipment constraints. Due to the dynamic and fluid nature of the ETTC mission, the scheduling processes of the RSD follow a cyclical pattern of initial scheduling followed by multiple changes which precipitates rescheduling. Quite often, even before the finishing touches are applied to an initial schedule, unforeseen changes make that schedule either difficult to execute or altogether invalid. Because these scheduling changes can become cumbersome and time consuming, the second objective of this research is to decrease the man-hours necessary to initially schedule RSD resources.

1.4. Range Support Division Scheduling Process

Jobs are usually assigned to InDyne by a 46th Test Wing Test Engineer (TE) in the form of a TO. InDyne in turn assigns these TOs to their division responsible for the majority of the work. The majority of the TOs received by InDyne are assigned to the RSD. TOs assigned to the RSD typically involve the construction or repair of range targets.

Jobs are also assigned to RSD via IWOs by other divisions of InDyne. This usually happens when another division is tasked to support a job but lacks the personnel, expertise or equipment required to complete a portion of the work. Therefore, IWOs are usually assigned to RSD for one of two reasons: 1) an InDyne division is assigned a job which requires RSD resources, or 2) an InDyne facility requires building maintenance.

TOs and IWOs are formatted with the information required to complete each job which include: work title, description of work, need date, and AAC priority number. Each job has a unique AAC priority code. AAC priority codes are only used to break ties when the simultaneous requirements of multiple jobs exceed available resources. AAC priority codes range from one to a thousand with the lowest number receiving the highest

priority. Generally, jobs assigned to RSD have AAC priority codes in the hundreds and above. Occasionally, RSD will receive a job with a priority code in the teens or single digits; those jobs will receive whatever resources are required to complete them.

The RSD supervisor assesses new TOs and IWOs to determine an estimate of the cost, manpower, equipment, and supplies required to complete the job. Once the RSD supervisor determines the resources required for each job, he assigns management responsibility of the job to one of RSD's six teams based on which team is responsible for the majority of the work. If resources from multiple RSD teams are required, the supervisor will create additional intra-RSD IWOs which task one or more of the five remaining teams to assist the team with the management responsibility. The team assigned with management responsibility coordinates with the other tasked RSD teams to complete the job. Therefore, the scheduling of team resources to specific RC and BM jobs is not directly managed by the RSD supervisor.

The DMS schedule is produced by the 46th TW. As explained previously, jobs associated with the DMS schedule are the RSD's highest priority. On a weekly and daily basis, the RSD supervisor schedules manpower and equipment to fulfill DMS jobs. Additionally, the RSD supervisor schedules his resources to work on TOs or IWOs that warrant special attention that day. If not tasked for DMS or the daily priority jobs, the remaining RSD resources are freed to their respective team leads to be scheduled as required to complete the highest priority TOs and IWOs within their purview. Therefore, outside of DMS or other high priority jobs, the weekly RSD scheduling of resources is delegated to team leads. This unwritten weekly RSD schedule of TOs and IWOs which is produced following DMS tasking is the schedule of interest for this research.

1.4.1. Purpose

The purpose of RSD's current scheduling process is to facilitate the completion of all assigned TOs and IWOs prior to their due dates.

1.4.2. Assumptions

TOs and IWOs are the basic scheduling units. The amount of schedulable units is primarily constrained by the deadline, range availability, available man-hours, and heavy equipment. Range availability is a product of the 46th TW Operations Order. RSD personnel are able to work on a site only when there are no ongoing missions. For this reason, RSD personnel rarely have unlimited or uninhibited access to a job site. Additionally, most of the job sites are at least one hour travel each way from Eglin (daily starting/ending point). Therefore, RSD scheduling rules assume that each worker will be productive no more than six hours per day. Although each job has unique material requirements, the materials are purchased as they are needed and are therefore not considered as constraining in this study. Finally, preemption is allowed in the schedule. It is unusual for other than minor jobs to be completed without preemption.

1.4.3. Resources and Constraints

The number of jobs RSD is able to complete each week is constrained by available range time, man-hours, and heavy equipment. The job deadlines are an important constraint in the scheduling process but are defined by the customer, independent of each other, and not a function of RSD’s scheduling process. If it is determined that a deadline will not be met, the RSD supervisor will contact both the customer and the 46th TW to request an extension, although this rarely happens.

1.4.3.1. Range Time

There are over three dozen range sites on the ETTC. RSD must have exclusive use of a range (work site) in order to accomplish required work. RSD does not usually have priority use of ranges for which they must accomplish work. Instead, they must wait for everyone else to make their requests. They are forced to find work on ranges they can gain access to on any given day; therefore, they must remain flexible. Every week, the 46th TW schedules range times according to user requests and 46th TW needs. Within that schedule, RSD is assigned to support ongoing missions at various times before, during, and after the mission time. It is RSD's primary mission to accomplish these jobs.

RSD's team leaders and supervisor attempt to maximize the amount of time they are active in productive work. Therefore, they attempt to limit the amount of travel time to and from work sights (ranges). In that respect, the team leaders will not schedule a job at a worksite where they are not able to get at least four hours of work done, assuming the job requires at least four hours of work.

1.4.3.2. Man-hours

The current number of workers on each of RSD's six teams are: Electricians (eight), Carpenters (six), Surveyors (four), Welders (four), Heavy Equipment Operators (eight), and Building Maintenance (five). Workers are rarely scheduled to work alone. For safety reasons, a minimum of two workers from a team are scheduled to each job. The workers are also very versatile in that they can work outside their team. For example, an electrician may assist a carpenter when there is no electrical work to be done.

RSD personnel work a straight work schedule with no scheduled breaks. The baseline schedule for each worker consists of nine hour work days Mondays through Thursdays with an eight hour work day on Fridays. Every worker has Saturdays and Sundays off as well as every other Friday off. Fifty percent of the workforce alternates Fridays off every week. Although overtime is discouraged unless mission essential, the RSD supervisor is allowed to schedule workers up to a maximum of twelve hours per day and sixty hours per week. The InDyne General Manager must approve scheduling workers in excess of those amounts. If need be, workers can be scheduled up to sixteen hours per day and seventy-two hours per week. It is InDyne's policy to allow a minimum of eight hours off between shifts. On special occasions, RSD will work on Saturday and/or Sunday to complete a project. See Appendix B for InDyne's maximum work time policy.

1.4.3.3. Heavy Equipment

RSD manages and operates government owned heavy equipment for use on their construction and maintenance missions (see Appendix D. List of Resources.) Some pieces of heavy equipment require specific pieces of other multi-use equipment for transportation or operation, and therefore must be scheduled together. For example, cranes and bulldozers usually require transportation to a job site on a trailer which must be pulled by a tractor. Although the type and number of these resources is finite, RSD does have the ability to temporarily rent extra or specialized equipment as needed. Additionally, there are instances when certain jobs are either too cumbersome or too technical for the RSD to accomplish indigenously. In those instances, RSD will

subcontract the job out to a company that is better equipped or capacitated to accomplish the task.

1.4.4. Current Methodology

The scheduling process begins when the weekly DMS schedule is received by the RSD supervisor. A tentative schedule of DMS tasks is published each Thursday by the 46th TW. From this, a tentative RSD schedule for the following week is produced. This schedule is updated three days prior to the day of execution. The actual execution schedule is created with the release of the AAC Operations Order (OP ORD) which is released one day prior to the day of execution. Following the construct above, an initial weekly schedule for the RSD is produced by the supervisor no later than the preceding Friday from which a daily schedule is created one day prior to the day of execution.

As each daily schedule is produced at the end of the previous day, the RSD supervisor allocated RSD resources from each of the six teams to fulfill DMS jobs that are tasked via the Op Ord. Additionally, the RSD supervisor schedules RSD resources for high priority RC or BM jobs. The list of high priority RC or BM jobs change daily. Often, items are added to this list based on external inputs from the 46th TW, from InDyne leadership, or from Range Safety. For example, any time a warning light burns out on a tower, for safety purposes the job of replacing the bulb automatically becomes a high priority job.

After all necessary resources are allocated to support DMS and high priority TOs and IWOs, the remaining resources are released to individual RSD team leads to utilize according to their priorities. Each remaining TO and IWO has unique requirements in terms of range, manpower, equipment required, and precedence. For instance, one job

may require that surveyors mark a job site before another job with heavy equipment operators can construct a target. All of the coordination required to accomplish TOs and IWOs are handled at the team leader level with overall management of resources handled at the RSD supervisory level.

Jobs are scheduled based on scheduling priority and customers need date. As previously stated, DMS jobs have the highest priority. All remaining RC and BM jobs associated with active TOs and IWOs are processed according to need date. When resources are constrained, ties for jobs with matching priority and/or need date are broken with the AAC priority code.

1.5. Preview

The remaining chapters contain a detailed explanation of the research project's methodology and conclusions. Chapter 2 describes the literature reviewed during the project. In Chapter 3, the research assumptions, scheduling rules, and constraints are defined. It also contains a detailed description of the methodology used to generate feasible solutions. Results and analysis are presented in Chapter 4. Finally, Chapter 5 lists the conclusions and recommendations of the research project.

Chapter 2. Literature Review

2.1. Resource Constrained Project Scheduling Problems

The purpose of this research is to create a scheduling tool for a Resource Constrained Project Scheduling Problem (RCPSP). RCPSP's are a class of scheduling problems that attempt to schedule activities of either a known (deterministic) or variable (stochastic) duration in a defined sequence given a finite amount of resources. RCPSP can formally be defined as a combinatorial optimization problem (Artigues, 2008: 21). Typically, the length or "span" of a schedule is optimized, more specifically minimized, such that resource and precedence constraints are adhered to. This research focuses on deterministic RCPSPs with precedence constraints where preemption is allowed. Multiple methods have been developed to solve such problems.

The next section of this chapter provide information on some basic concepts in scheduling theory. A discussion of computational complexity is included to explain the applicability of different approaches described later in the chapter. The algorithm created during this research applies the greedy principles and is a heuristic technique for solving a RCPSP. Alternative methodologies are discussed in order to cover other solution methods investigated during this research. Finally, in order to provide a theoretical basis for later chapters, an explanation of the heuristic approach used to create the scheduling algorithm of this research is presented at the end of the chapter.

2.2. Scheduling Theory

Project scheduling is a decision making process that is commonly found in all types of organizations both large and small, from private industry to government. Although scope and objectives change, the problem still lies in how to schedule activities

given technological constraints. Technological constraints are typically either precedence or resource related. Activities and constraints can take on many forms, from the typical job shop with a single machine, one type of resource, and one product; to a group of specialized workers with disparate tasks and multiple resources. Many analytical methods have been created to solve the range of possible problems.

Broadly stated, there are three common analytical methodologies used to model a scheduling process: linear programming, enumeration, and heuristic techniques. Whereas linear programming and some enumerative solution methods are used to find optimal solutions, should one exist, many enumerative and heuristic approaches are used to quickly find better solutions to problems that would have otherwise been found without applying any analytical technique at all. Project scheduling problems or "instances" are generally classified by the following taxonomy:

n/m/A/B where:

n ≡ Number of jobs to schedule.

m ≡ Number of machines to process the jobs.

A ≡ Flow pattern within the job shop.

B ≡ Performance objective of the schedule (e.g. minimize the lateness of job completion times).

For example, $n/2/O/L_{max}$ would denote a schedule for an n job, 2 machine, open job shop where the objective is to minimize maximum lateness.

Furthermore, each RCPSP is further defined by the inter-relationship of its activities, durations, precedence relations, resources, and demands (Artigues: 22). All possible feasible solutions obtained through the combination of these attributes define the

solution space from which the schedule is ultimately created. Additional attributes of a schedule may include task due dates and preemption. Preemption is a scheduling rule that permits jobs to be paused once begun to allow for the start of another job. Preemption and deadlines increase both the solution space and complexity of finding an optimal solution via any method, as is the case in this study.

2.3. Computational Complexity

The computational complexity of a problem is defined as the difficulty of finding a optimal solution given a particular solution technique. As stated by Parker and Rardin, "…complexity seeks to classify problems in terms of the mathematical order of the computational resources required to solve the problems via digital computer algorithms." (1982). Problems are divided between two general categories of complexity, those that have algorithms that can be solved to optimality in Polynomial time (P) and those that lack such algorithms which are called Non-deterministic Polynomial (NP). Examples of problems with polynomial time algorithms include most assignment, covering, and simple linear programming problems, to name a few. If the problem is of small to moderate size, these algorithms can be solved to optimality in a sensible amount of time on digital computers because their algorithms have been proven to solve to optimality in polynomial time. If a class of problems is considered NP, there exists no algorithm which will either solve for feasibility (decision problem) or check optimality (recognition problem) of the problem in Polynomial time; rather the worst-case time required to solve these problems to optimality is more closely approximated by some exponential function, meaning the solution time required to find an optimal solution increases exponentially as the number of variables increase (French, 1982: 146). For example a particular algorithm

may solve a P class problem with thirty variables in .00003 seconds on a digital computer, while an NP class algorithm for a different problem of the same size (thirty variables) would take 8.4×10^{16} centuries to find an optimal solution on that same computer (French: 141). Furthermore the same NP problem may take only 3.6 seconds to solve to optimality when only ten variables are used instead of thirty.

As shown above, it is important to understand the practical types of solution methods that exist for a particular scheduling problem. As far as RCPSPs are concerned, many instances belong to a sub-class of NP called NP-hard or NP-complete. These types of problems are very hard to solve to optimality and tend to lend themselves to heuristic methods, which find feasible solutions in a more reasonable amount of time, although they often sacrifice optimality to do so. The RCPSP of this research is characterized as $Om//L_{max}$, which is an open job shop with *m* separate machines that has an objective to minimize maximum lateness. Mathematically, that particular type of problem is classified as strongly NP-Hard (Pinedo, 2008: 605).

2.4. Deterministic Methodology

Solutions to RCPSPs are found a number of ways. Four commonly used solution techniques are linear programming, enumeration, constraint programming, and heuristic algorithms.

2.4.1. Linear Programming

Linear Programming (LP) algorithms are a class of solution techniques that find optimal solutions to various scheduling problems. As its name suggests, LP algorithms are comprised of a series of equations of linear combinations of decision variables. Methods include traditional Linear Programming (LP) where decision variables belong to

the set of real numbers, Integer Programming (IP) techniques where decision variables are restricted to integer values, Mixed Integer Programming (MIP) where some decision variables are continuous and the rest are integer, and binary decision variables where decision variables are limited to the values 0-1. Figure 2.1 shows the classic formulation of a MIP with all three types of variables.

$$Max\ z = \sum_{j=1}^{n} c_j x_j \qquad [2.4.1]$$

subject to:

$$\sum_{j=1}^{p} a_{ij} x_j + \sum_{j=p+1}^{n-1} a_{ij} x_j + \sum_{j=n} a_{ij} x_j \left\{ \begin{matrix} = \\ \leq \\ \geq \end{matrix} \right\} b_i \qquad [2.4.2]$$

$$x_j \geq 0 \text{ and integer for j = 1,...,p} \qquad [2.4.3]$$

$$x_j \geq 0 \text{ for j = p+1,...,n-1} \qquad [2.4.4]$$

$$x_j = \begin{cases} 0 \\ 1 \end{cases} \text{ for j = n} \qquad [2.4.5]$$

Figure 2.1 MIP Formulation

Figure 2.1 depicts the typical format of a LP with a single objective function [2.4.1] and constraint equations [2.4.2] through [2.4.5]. The objective function is a linear combination of the decision variables multiplied by some type of cost coefficient and the objective will seek to either maximize or minimize its value. Each of the "i" constraint equations would represent a single precedence or resource constraint [2.4.2], or set limits on the value of the decision variables [2.4.3] through [2.4.5]. Each resource or precedence constraint can be either an equality, upper, or lower bound (b_i) as indicated by the column vector in equation [2.4.2].

LP algorithms will find an optimal solution if one exists. Quite often, LP formulations are used to solve simple problems with limited variables or simplified versions of more complex problems. In practice, too much fidelity may be lost by over simplifying a complex problem. Therefore, other solution techniques are often utilized to solve complex RCPSPs in order to avoid the solution times of NP algorithms described in the previous section.

2.4.2. Enumeration

Enumerative techniques are sometimes used to solve RCPSPs when it is determined that the LP formulation is either too complex to simplify or will take too much processing time to solve (NP). Enumeration algorithms are either explicit or implicit. An explicit enumeration algorithm solves for every possible combination of the decision variables. For each solution, the algorithm compares the objective function value to the reigning best solution in order to find the optimal solution once all possible solutions have been investigated. Usually, if a problem takes too much processing time to solve with some version of an LP formulation, then it will also take too much time to solve via explicit enumeration. Therefore, more eloquent algorithms have been discovered which solve only a portion of the total solution space, otherwise known as implicit enumeration. Some examples of implicit enumeration are branch-and-bound and branch-and-cut. Each of these techniques uses similar rules to logically explore or eliminate portions of the solution space, thus requiring only a fraction of the computing time required for explicit enumeration. Although this approach is applied to more complex problems than LP, IP, MIP, or explicit enumeration methods, it does not guarantee optimality of the solution like those methods (Murty, 1995: 363).

2.4.3. Constraint Programming (Pinedo, 2008)

Constraint Programming (CP) is similar to LP methods in that logical constraints are used to define relationships between decision variables. CP does not solve to optimality through the use of strict mathematical equations like LP methods; rather, it seeks to vary decision variables in accordance with a logically based process in order to produce a feasible solution. Whereas constraints in mathematical formulations are either linear or non-linear, constraints in a CP formulation can be more general in form. Examples of constraints may include logical, linear, non-linear, cardinality, or global constraints. Another significant difference between mathematical and CP formulations is that CP does not have to use an objective function; rather, a CP algorithm's objective is to just find a feasible combination of decision variables. Intelligent designs can reduce the number of permutations to run before an acceptable solution is reached. (Pinedo, 2008: 582) Figure 2.2 shows an example of a high order CP.

```
While not solved AND not infeasible DO
        consistency checking (domain reduction)
        IF a dead-end is detected THEN
                try to escape from dead-end (backtrack)
        ELSE
                select variable
                assign value to variable
        ENDIF
ENDWHILE
```

Figure 2.2 General CP Formulation (Pinedo: 582)

CP algorithms search through the solution space in a manner similar to that of branch-and-bound techniques. Each branch terminates at a node which represents an assignment of one variable, from which another bound can be made or not. At each

node, the combination of variables are tested for feasibility. If the combination of variables is deemed infeasible, then the algorithm follows "backtrack" rules which move out of the infeasible region and into a feasible one. Once all variables have been assigned a feasible value, the resulting schedule is compared to some threshold metric. If the schedule meets or exceeds that metric, then the schedule is complete. Additional CP methodologies can be used to seek improvements to the schedule. (Pinedo, 2008: 587)

2.4.4. Heuristics (French, 1982)

Heuristic methods should not be used if one of the optimal solution methods discussed previously is computationally feasible (French, 1982: 156). That being said, heuristic approaches to scheduling problems are widely used throughout industry. The reason being that real world industrial problems usually involve a complex set of scheduling rules that are difficult to code mathematically, not to mention the processing time required to solve these problems once those rules are coded. Heuristic methodologies seek to "...use our knowledge and experience to find a schedule which, if not optimal, may at least be expected to perform better than average", and do it in a reasonable amount of time (French, 1982: 155). In much the same way CP approaches use constraints to enable searches for feasible solutions, heuristic approaches use scheduling rules to build a feasible schedule. These rules can be formulated as logical constraints, mathematical expressions, or even mini-LP subroutines. Heuristic techniques are not tied to one methodology; rather, they are a collection of scheduling rules used to create schedules otherwise built by hand. The rest of this chapter covers general heuristic scheduling approaches as well as detailing methods utilized in this study.

2.5. Heuristic Methods

Some heuristic methodologies are generically applicable to in a broad range of scheduling problems. Conversely, some algorithms are applicable for only specific cases. For this study, a general schedule generation technique was applied. The process used to create the heuristic scheduling algorithm for this study is comprised of two parts: creation of a priority list and the application of a scheduling scheme to the priority list. Although not used in this study, it is not uncommon to attempt to make a schedule better by applying metaheuristic improvement algorithms to a feasible schedule after it is created.

2.5.1. Greedy Algorithm

The heuristic approach used in this research applies greedy principles. A solution method is said to be a greedy method if it has the following features: incremental, no-backtracking, and greedy selection (Murty, 1995: 414). An incremental solution method is one that is created from a subset of elements in which the algorithm sequences through the subset until a complete solution is reached. No-backtracking refers to the manner in which decisions about the solution set are made. A heuristic method with a no-backtracking feature does not revise a decision once it is made, whether it be inclusion of an element to the solution set or not. The greedy selection feature implies that the best available element from a selectable set is used for the next stage in the process. The criterion for selecting the best element is problem specific but may include attributes such as least cost, highest profit per unit, or some other user defined priority scheme.

2.5.2. Priority List Scheduling

Priority list scheduling is the most commonly used technique in industry because it is intuitive to understand and implement. Additionally, it is employed in many

commercial scheduling packages because they are computationally efficient (Kolisch, 1996). Priority lists can be thought of as a prioritized list of activities used as input for a schedule generator. In relation to the three greedy features covered in the previous section, priority lists are commonly used as the engine for the greedy selection feature.

Priority lists are created by prioritizing a list of events using dispatching rules. Dispatching rules can be either static or dynamic (Pinedo, 2008: 372). Dynamic rules are time dependent while static rules are not. An example of a dynamic dispatching rule is minimum slack. Minimum slack (MS) is the difference between the time an event is due and the current time plus the remaining event processing time. Thus, as time progresses, an event's minimum slack decreases, which may alter its position on a priority list. Other examples of dispatching rules are First Come First Serve (FCFS), Service in Random Order (SRO), and Shortest Processing Time. Yang studied common dispatching rules used for both deterministic and stochastic processes. Yang found that five dispatching rules are statistically better at producing the shortest mean project completion times, the most number of shortest project completion times and the least number of longest project completion times when applied to a variety of scheduling instances (Yang, 1998). Two of these rules, Minimum Slack and Greatest Cumulative Resource Requirement (GCRR) are used in the algorithm produced during this study. Once a priority list is created, a schedule generation scheme is used to create the first instance of the schedule.

2.5.3. Schedule Generation Schemes

Two of the oldest and most widely used heuristics for the RCPSP are the serial and parallel generating schemes (Kolisch, 1982).

2.5.3.1. Serial vs. Parallel (Kolisch, 1982)

Serial scheduling schemes select and schedule one activity at a time. In this scheme, a single event is selected from the set of un-scheduled events that belong to a Decision set (D). After the event is scheduled, it is placed in the set of scheduled events (S). This in turn may bring more events into the decision set if the event just scheduled is a predecessor of those events. This process is repeated until all events are placed in S.

Parallel scheduling schemes are similar in that only one event is scheduled at any one time. One event is selected from the decision set in accordance with the same dispatching rules as well. When scheduled, events are added to the Active set (A), meaning they have been scheduled but are not completed by the current scheduling time. As the current schedule time is advanced, events that reach completion are placed in the Completed set (C). Current schedule time is advanced to the earliest completion time of all activities in A.

Kolisch's study of serial and parallel scheduling schemes concluded that the serial scheme performs better when applied to large problems or when resources are only moderately constrained (1982). Additionally, it was proven that serial schemes produce active schedules while parallel schemes produce only non-delay schedules. In an active schedule, events cannot be scheduled any earlier without violating technological constraints, which is referred to as a left-shifted schedule. Non-delay schedules sequence activities such that no machine is kept idle when there is an event it could start processing (French, 2008: 157). The significant difference between active and non-delay schedules is that the solution space for an active schedule will contain the optimum schedule while a non-delay schedule may not.

2.5.3.2. Single Pass vs. Multi-Pass

Both serial and parallel methods can be applied in either a single or in multiple passes. The single pass heuristic is described above. A multi-pass heuristic iteratively applies either a serial or parallel scheduling scheme, each time altering the dispatch rule used. Each time a pass is completed it creates a feasible schedule which is compared to existing feasible schedules of which the best one is selected (Kolisch, 1982). Another variant of the multi-pass approach is called the forward-backward pass heuristic. The forward pass applies a serial scheme thereby producing an active schedule (left shifted), then the backward pass solves for the mirror problem thereby producing a right shifted schedule. In this manner, the heuristic repeats these iterative steps until no improvements can be made from one shift to the next (Artigues, 2008: 91).

2.5.4. Interchange Metaheuristics (Pinedo: 378)

Interchange metaheuristics are an improvement technique that is applied to schedules once they have been built using methods just discussed. The two methods described in the following sections are called local search procedures. Local search procedures do not guarantee an optimal solution. Rather, they seek to find a better schedule through alteration of the current one (base schedule). The base schedule is modified by some sort of predefined process, usually a pair-wise interchange of two or more events within the schedule. A "neighbor" schedule is created once an altered schedule is proven feasible. Each neighbor created from the base schedule is compared to some predefined metric or acceptance/rejection criteria. The best neighbor is selected as the new "base" schedule and the process is repeated until it reaches its termination criteria (Pinedo, 2008: 378). The difference between methods lies in the manner in which

new neighbors are created, accepted, and the criteria used to end the metaheuristic process. Below is a description of two common techniques called tabu search and simulated annealing.

2.5.4.1. Tabu Search

In tabu search, deterministic rules are used to approve new neighbors. As the algorithm progresses through a solution space, or tree, a tabu list is kept which stores past interchanges in their reverse sequence. Tabu lists can vary in length, usually storing no more than five to nine reverse interchanges at any one time. The algorithm prohibits returning to previous solutions through the use of the tabu list. The goal of the algorithm is to search for the best solution while avoiding local minimum. The tabu search will use a deterministic procedure to select the new base schedule from its neighbors, which may actually be a worse solution than the current best. In this way, the algorithm attempts to back out of local minimum in order to find a better solution later on in the search (Pinedo, 2008: 384).

2.5.4.2. Simulated Annealing

Simulated annealing is similar to a tabu list algorithm, but instead it uses stochastic methods to approve new base schedules from within a neighborhood. Typically, a selection probability will be assigned to each neighbor in the neighborhood. The worst neighbors will be assigned lower probabilities of selection as the algorithm progresses. In turn, by assigning progressively better probabilities to better solutions, the algorithm will tend to back out of and local minimums early in the process and hone in on better schedules later in the process (Pinedo: 380).

2.6. Summary

The Resource Constrained Project Scheduling Problem (RCPSP) tends to be difficult to mathematically solve to optimality. Even problems of moderate size and dimension can take an exorbitant amount of time to solve using analytical techniques such as linear programming. Subsequently, alternate methods are often used to create feasible schedules for many RCPSPs. Some alternate methodologies like Branch-and-Cut may still find optimal or nearly optimal solutions by implicitly enumerating the possible solution space of the problem. In large industrial applications though, even these types of methods can prove to be intractable. Heuristic algorithms are an alternate methodology which will quickly find good and sometimes optimal solutions to complicated RCPSPs.

Greedy heuristic methods that use either a serial or a parallel schedule generating scheme are two commonly used methods to solve difficult RCPSPs. Heuristic approaches are easy to conceptualize and adapt to industrial applications; therefore, they are found in many industrial software packages. Much research has been devoted to creating heuristic algorithms that solve different types of RCPSPs. Both constructive algorithms, like serial or parallel, and improvement algorithms, like tabu search or simulated annealing, can be adapted and applied to almost any kind of problem.

Chapter 3. Methodology

3.1. Overview

This chapter outlines how the heuristic methods discussed in Chapter 2 are applied to the RCPSP described in Chapter 1. In this study, a greedy heuristic method is used to generate a schedule for a RCPSP. The heuristic creates a prioritized list which is used to serially schedule jobs in a single pass. This heuristic will be referred to as a Single-Pass Serial Schedule (SPSS) algorithm throughout the rest of this paper. A detailed explanation of the SPSS algorithm used to generate a feasible schedule is provided. In section 3.2, the algorithm's parameters and variables are defined. Section 3.3 lists the assumptions made to simplify the problem enough to enable the creation of a workable algorithm. The next seven sections break up the code into logical segments to facilitate a more detailed explanation of the algorithm's processes. These sections are titled: SPSS routine, prioritize subroutine, insertion subroutine, range test, manpower test, equipment test, and metric subroutine. The SPSS routine is the top-level routine which calls on both the prioritize, insertion, and metric subroutines. The range, manpower, and equipment tests are all contained within the insertion subroutine. Figure 3.1 depicts the hierarchy of the (sub)routines in addition to the input and output of each (sub)routine/test. Appendix D lists the full algorithm start to finish.

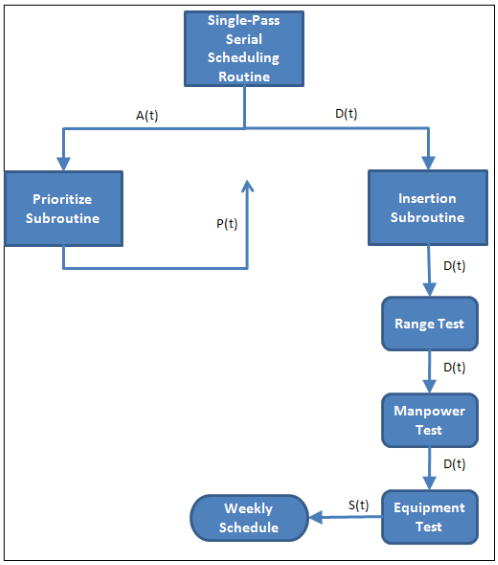

Figure 3.1 Routine Hierarchy

3.2. Parameters and Variables

The first step in outlining any analytical approach to problem solving is to define the parameters and decision variables used. The parameters used in this algorithm are related to time or are functions of time. "t", defined on line [3.2.1], represents the current scheduling day. WW on line [3.2.2] is a user defined parameter which allows decision makers the flexibility to alter the search parameters within the scheduling algorithm to reflect the week's work schedule.

t ≡ Current scheduling day (MM:DD:YYYY), t ∈ {(1:1:2009), (1:2:2009),...} [3.2.1]
t_o ≡ First day of the scheduled week (Monday)
WW ≡ Work Week, 2x5 matrix of starting (SW(t)) and ending (EW(t)) times for schedulable work during day t for the weekly schedule [3.2.2]

The example WW matrix below shows a 5 day work week for SW(t) = 0700 and EW(t) = 1500 for the first four days (t = 1, 2, 3, 4); and for the fifth day SW(5) = 0700 and EW(5) = 1400. Each number represents a sixty minute period (eg. "7" = 0700:00 to 0759:59). For example, eight hours are spanned between SW(5) and EW(5) (0700 to 1459), which is a typical Friday work schedule.

$$WW = \begin{pmatrix} 7 & 7 & 7 & 7 & 7 \\ 15 & 15 & 15 & 15 & 14 \end{pmatrix}$$

The following definitions of the decision variables are largely self explanatory. Set notation is used throughout the algorithm. Arrays and matrices are used when appropriate to store and manipulate related data.

$K_r(t) \equiv$ Array of available resources for each day t, $r \in \{R, M, E\}$ [3.2.3]
- $R(t) \equiv$ Matrix of range availability per period for each "SITE" and day "t"
- $M(t) \equiv$ Matrix of man-hours available per period for each "TYPE" and day "t"
- $E(t) \equiv$ Array of heavy equipment available for day (t)

$A(t) \equiv$ Active Set of arrays a_j for each job j in a scheduling day t, $j \in \{1, 2, 3, \ldots, J\}$ [3.2.4]
- $a_j \equiv$ (SWN, SITE, PRI, MH, WPH, TYPE, (EQP), PRED, SUC, DL, MS)
- $SWN_j \equiv$ Service Request or Work Order Number of job j
- $SITE_j \equiv$ Work site identifier, $\in$ {of all ranges}
- $PRI_j \equiv$ Priority number, $\in \{1, 2, 3\}$
- $MH_j \equiv$ Man Hours required for job j, $\in \{0, 1, 2, 3, \ldots\}$
- $WPH_j \equiv$ Workers Per Hour required for job j, $\in \{1, 2, 3, \ldots\}$
- $TYPE_j \equiv$ Type of worker required for job j, $\in$ {B, C, E, H, S, W}
  - B = Building Maintenance; C = Carpenter; E = Electrician;
  - H = Heavy Equipment Operator; S = Surveyor; W = Welder
- $(EQP)_j \equiv$ Array of Heavy Equipment required for job j, (see Appendix C.)
- $PRED_j \equiv$ Job that must be completed prior to the start of job j (Predecessor)
- $SUC_j \equiv$ Job that can only start once job j is completed (Successor)
- $DL_j \equiv$ Dead Line of job j, $\in$ {(MM:DD:YYYY)}
- $MS_j \equiv$ Minimum Slack of job j; $MS_j \equiv DL_j - (t + TPT_j)$ [3.2.5]
  - $TPT_j \equiv$ Total Processing Time $\equiv MH_j + \sum$MH of all Successors of job j

$P(t) \equiv$ Prioritized, set of arrays p_j (sorted elements of A(t)); $p_j \leftarrow a_i, \forall\ i, j$ [3.2.6]

$D(t) \equiv$ Decision, set of arrays d_j of schedulable jobs from P(t); $d_j \leftarrow p_j, \forall\ j$ [3.2.7]

$S(t) \equiv$ Scheduled, set of arrays s_j of scheduled jobs [3.2.8]
- $s_j \equiv$ (SWN, SITE, TYPE, WPH, (EQP), START, END)
- $START_j$ = Start time of work for job j; END_j = End time of work for job j

$WT_j = MH_j/WPH_j$; Work Time, hours required at $SITE_j$ to complete job j, $\in \mathbb{R}$ [3.2.9]

Of note, sets A, P, D, and S above are used repetitively within and between days to facilitate a SPSS generation scheme. The set of all scheduled jobs for each day, S(t)

on line [3.2.8], contains all the scheduled jobs on a particular day in array format. The cumulative elements of this set, for each of the five days, is the end product (schedule) of the algorithm.

Line [3.2.5] is the minimum slack of job j. Minimum slack was covered in Chapter 2 as a key dispatching rule. MS is a function of a jobs Dead Line (DL), the current scheduling day (t) and the Total Processing Time (TPT). The TPT of job j is calculated by adding the remaining man-hours required for job j plus the summation of the man-hours of all the successors of job j. TPT is analogous to the Greatest Cumulative Resource Requirement (GCRR), also discussed in Chapter 2.

Work Time (WT) [3.2.9] is the time required on a particular job site. For example, if a job requires twenty man-hours of electrician work to be done by two electricians, then WT = $\frac{\text{MH}}{\text{WPH}} = \frac{20}{2} = 10$ hours on site.

3.3. Assumptions

The RSD schedule is difficult to solve to optimality. Table 3.1 lists the assumptions that are made about the original process to make it easier to model within the context of the SPSS algorithm.

1	The scheduled work week will be comprised of one shift per day of nine-hours Monday through Thursday, and one eight-hour shift on Friday. No overtime will be modeled.
2	Available man-hour resources are aggregated for each team. This assumption discounts the time required to move between job sites within a single day.
3	Individual pieces of heavy equipment will only be scheduled to one job per day. This may limit some jobs from being scheduled on a day they are otherwise capable of being started.
4	Only six hours of production per worker will be counted per day against a jobs total required man-hours regardless of the number of hours scheduled.
5	Preemption of jobs is allowed.
6	No jobs requiring TDY will be considered in this algorithm. They will have to be scheduled outside this procedure and the resources subtracted from available resources by user input.
7	Assume one hour of travel each way to each site.
8	Jobs are assigned a priority number based on the type of work to be done. Some jobs have predecessors, successors, or both. For this algorithm, it is assumed that successors will be assigned the same priority number as their predecessors.

Table 3.1 Algorithm Assumptions

3.4. Single-Pass Serial Scheduling Routine

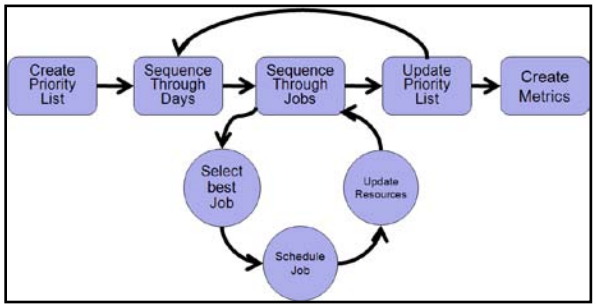

Figure 3.2 SPSS Routine Flow Chart

Figure 3.2 displays how jobs flow through the SPSS routine, which is covered in the following sections. The algorithm is written in pseudo code. To facilitate explanation of the algorithm's processes, the code is broken up into smaller segments throughout this and the next six sections. The SPSS routine is outlined in this section.

Single-Pass Serial Scheduling Routine

START
$n = 0$ [3.4.1]
$t = t_o$ [3.4.2]
$A(t) = P(t) = D(t) = S(t) = \emptyset$ [3.4.3]
RUN Prioritize Subroutine [3.4.4]

The first step in the top-level routine initializes "n" to zero [3.4.1]. "n" is used throughout the code to logically increment through each job "j" as the algorithm schedules within a day "t" (incremental feature of greedy methods). "t" is initialized to the first day of the scheduled week on line [3.4.2]. "t" is used as a counter to ensure the algorithm runs through the routine five times (i.e. an entire five-day work week). Line [3.4.3] initializes all of the sets to the empty set (∅). The operations of the Prioritize subroutine on line [3.4.4] are covered in the next section.

WHILE $t \leq (t_o + 5 \text{ days})$ DO [3.4.5]
BEGIN
  WHILE $n \leq J$ DO [3.4.6]
  BEGIN
    IF $d_n \in D(t)$ THEN [3.4.7]
      RUN Insertion Subroutine
    ENDIF
    $n = n + 1$ [3.4.8]
    $D(t) \leftarrow \{p_n \mid \text{“}PRED_n\text{”} = 0 \text{ OR “}PRED_n\text{”} = \text{“}SWN_i\text{” with “}MH_i\text{”} = 0\}$; [3.4.9]
      $\forall i, i \in \{1, 2, 3, \ldots, J\}, i \neq n, d_n \leftarrow p_n$
    $P(t) \equiv \{p_j \mid \text{“}SWN_j\text{”} \notin D(t)\}, \forall j$ [3.4.10]
  END WHILE

The outer WHILE loop on line [3.4.5] sequentially steps through each day "t" of the scheduling week. The inner WHILE loop [3.4.6] sequences through each job j currently in the set P(t) for the current day t. Line [3.4.7] determines if the current job is a member of the decision set, D(t). On the first pass through this IF statement, "n" will equal zero and D(t) will equal the empty set (∅); therefore the IF statement will be FALSE. On subsequent passes, line [3.4.7] will determine if prior passes through the inner WHILE

loop have placed the n^{th} job of the prioritized set P(t) in the decision set D(t). Keep in mind that D(t) is the set of schedulable jobs. If this IF statement is TRUE, then the algorithm will run the insertion routine which will attempt to find a valid time for the n^{th} job in day t, given available resources. The Insertion subroutine is covered in Section 3.4.2. After the insertion routine, "n" is incremented to the next job number on line [3.4.8]. Line [3.4.9] updates the decision set D(t) with the n^{th} job of the priority set P(t) if that job has no predecessor (PRED = 0) or if the n^{th} job has a predecessor with "MH" = 0 (i.e. predecessor job has been completely scheduled.) "$d_n \leftarrow p_n$", on line [3.4.9], indicates the elements of the array p_n will map to d_n if these conditions are true. Line [3.4.10] is the last line of code in the inner WHILE loop. Its function is to update the priority set with all elements currently in P(t) that are not also a member of the decision set D(t). This operation will remove the n^{th} array from the priority set P(t) if it is currently in the decision set D(t) because it met one of the conditions of the prior operation [3.4.9]. This inner WHILE loop will continue until all the jobs have been sequenced through it (n = J) and is terminated when n = J + 1.

n = 0
t = t + (1 day) [3.4.11]
$A(t) \leftarrow P(t\text{-}1) \cup \{d_j \in D(t\text{-}1) \mid \text{“}MH_j\text{”} > 0\}, \forall j, a_j \leftarrow p_j, a_j \leftarrow d_j$ [3.4.12]
RUN Prioritize Subroutine [3.4.13]
$D(t) \leftarrow \{p_j \in P(t) \mid \text{“}SWN_j\text{”} = \text{“}SWN_i\text{”} \text{ of } d_i \in D(t\text{-}1)\}, \forall i, j$ [3.4.14]
$P(t) \equiv \{p_j \mid \text{“}SWN_j\text{”} \notin D(t)\}, \forall j$ [3.4.15]
END WHILE
Metrics Subroutine
STOP

Following completion of the inner WHILE loop the algorithm reinitializes "n" and increments "t" for the next day of scheduling on line [3.4.11]. Before the end of the SPSS routine, the algorithm does four additional operations to prepare itself for the next

increment (next scheduling day). First, on line [3.4.12], it initializes the current days active set A(t) with the prior days priority set P(t-1) plus any elements of the prior days decision set D(t-1) that have not been completely scheduled. In this way, the algorithm places all remaining unscheduled or unfinished jobs in the next day's active set A(t). Then line [3.4.13] reruns the Prioritize subroutine which creates a new priority set P(t) from A(t). The operation on line [3.4.14] places all elements of the prior days decision set D(t-1) into the new decision set D(t) that were not completely scheduled the day prior. This is done so that the recursive operations contained in the inner WHILE loop can be reused each schedule day without creating new logic to populate the new decision set D(t). Finally, line [3.4.15] erases arrays from P(t) that were just mapped to D(t) exactly the same way it was done on line [3.4.10] earlier. The next five sections cover in more detail the operations of the prioritize, insertion, and metric subroutines.

3.4.1. Prioritize Subroutine

The prioritize subroutine creates the priority set P(t) from the active set A(t). The priority set is a set of all schedulable arrays listed in order of the priority rules discussed below. This priority list is the greedy selection feature of this algorithm in that it enables the insertion subroutine to select the best available job as the next scheduling candidate. The first operation the prioritize subroutine performs (line [3.4.16]) is to build the set A(t), which is a set of arrays of all active jobs to be scheduled within the RSD for the week. This process was not explicitly modeled in this study. The number of jobs in the active set (J) is then calculated by setting J equal to the cardinality of the active set, |A(t)|. The prioritize subroutine rank orders all active jobs according to three dispatching rules:

increasing order of Priority (PRI), increasing order of Minimum Slack (MS), and decreasing order of Total Processing Time (TPT).

If $t = t_o$ THEN
  Create J arrays of set A(t) from job and user input databases [3.4.16]
  Number the arrays of a_j sequentially, j = 1, 2, 3, …
ENDIF
J = |A(t)|
Sort the J elements of A(t) by: [3.4.17]
  1) Increasing order of priority number (PRI_j)*
     *A Successor will have same priority number as their predecessor
  2) Then by increasing order of Minimum Slack (MS_j)
  3) Then by decreasing order of Total Processing Time (TPT_j)
Place ordered elements of A(t) in P(t), sequentially renumber the J elements of P(t);
  $p_j \leftarrow a_i$, i = 1, 2,..., J, j = 1, 2, 3, …, J [3.4.18]

Priority number is a user defined attribute of each job which signifies the importance the organization places on completion of that particular task. RSD's priorities are outlined in Chapter 1. The MS and TPT dispatching rules ensure that the most difficult jobs (most processing time) to schedule are given priority over easier ones. Specifically, big jobs (greater man-hours) that have the smallest difference between their deadline and the current date are given the highest priority. For example, after the jobs have been ordered according to their priority number, they will be ordered according to smallest MS within each priority. Since all jobs with a predecessor have the same priority as their predecessor, this operation in effect rank orders jobs in order of their stated precedence. If there are jobs of the same priority with the same MS value, the algorithm will place priority on the one with the greatest amount of work left to be done (TPT). This prioritizing process can easily be accomplished with standard Microsoft Excel functions along with some additional Visual Basic Application (VBA) coding. The final line of the algorithm, [3.4.18] is used to sequentially renumber the elements p_j of the

priority set P(t) to enable the sequential operations of the SPSS algorithm that was presented in section 3.4.

3.4.2. Insertion Subroutine

A job j is processed by the Insertion subroutine once it is determined to be a member of the set D(t). This routine performs the necessary operations required to identify valid times to schedule job d_n. A comparative test is conducted for three different resources: range availability, man-power availability, and heavy equipment availability. Three tests, one for each resource, are embedded within the Insertion subroutine. They are called range availability test, manpower availability test, and equipment availability test.

The first operation of the insertion subroutine on line [3.4.19] calculates Work Hours (WH) for each job j. This operation finds the number of range periods that will need to be scheduled for each job. For example, if an Electrician job has sixteen hours of processing time remaining (MH) and the job requires three Electricians per hour, then the number of range periods required to finish the job equals:

$$\text{HOURS} = \left\lceil \frac{16}{3} \right\rceil = \lceil 5.33 \rceil = 6.$$

$$\text{For job } d_n\text{: SET HOURS} = \left\lceil \frac{MH_n}{WPH_n} \right\rceil \qquad [3.4.19]$$

The following three sections discuss the comparative tests performed within the Insertion subroutine starting with the range availability test.

3.4.3. Range Availability Test

Range Test
START
  FOR d_n Search R(t)(Row = "$SITE_n$") from (SW(t) + 1) to (EW(t) -1) [3.4.20]
  to find the maximum length string of consecutive free periods
    SET BG = First free period in string
    SET FS = Last free period in string
    WT = FS – BG + 1 [3.4.21]

The range availability test determines if there is sufficient time on day t to perform the required work on range "$SITE_n$" for job n, a member of the decision set D(n). The first operation the range test performs is to search within the range resource matrix R(t) to find consecutive free periods in the row labeled "SITE" and within the columns (SW(t) + 1) to (EW(t) -1). Only one event can be scheduled on a range at any one time, so this matrix is a 0-1 matrix. R(t) is defined as follows:

$$R(t) = \begin{cases} 1 \text{ if the range is not scheduled during a period} \\ 0 \text{ if the range is already scheduled during a period} \end{cases}$$

The algorithm searches between the periods specified by the matrix WW (Work Week). The WW matrix specifies the hours that manpower will be available to conduct work during the scheduling week (see equation [3.2.2]). Since a hour of travel time is needed to travel to and from every location, the algorithm only searches an hour after start of work (SW(t) + 1) to an hour before end of work (EW(t) – 1) (see line [3.4.20]). Three new variables WT (Work Time), BG (Begin), and FS (Finish) are used to keep track of the string length as well as the beginning and finish periods of the string. Figure 3.3 is used to illustrates this process.

	Period									
RANGE	7	8	9	10	11	12	13	14	15	…
SITE 1										
SITE 2	1	0	1	1	1	1	1	1	1	0
SITE 3										
SITE 4										
SITE 5										

Figure 3.3 Sample R(t) Matrix

The usual work shift per day is nine hours, except on Friday when it is eight hours. In the example in Figure 3.7, the values of SW(t) and EW(t) for the row "SITE 2" are "7" and "15" respectively, which represents a nine hour shift. The longest string of consecutive free periods is between periods "9" and "15", which equals seven consecutive hours. Since the algorithm limits the search to periods between (SW(t) + 1) and (EW(t) – 1), the actual values of BG and FS will be "9" and "14", respectively with WT equal to "6" (see equation [3.4.21]). These values are used throughout the rest of this test as well as the following tests to determine if there exists a feasible time to insert job n into the schedule.

```
    IF (HOURS < 4) AND (WT ≥ HOURS) THEN                    [3.4.22]
       (START = BG) AND (END = FS)
       GOTO Manpower Test
    ELSEIF (HOURS ≥ 4) AND (WT ≥ 4)                         [3.4.23]
       (START = BG) AND (END = FS)
       GOTO Manpower Test
    ELSEIF (HOURS < 4) AND (WT < HOURS)                     [3.4.24]
       RETURN NO SOLUTION
    ELSEIF (HOURS ≥ 4) AND (WT < 4)                         [3.4.25]
       RETURN NO SOLUTION
    ENDIF
 STOP
```

The lines of pseudo code above are a series of scheduling rules that when applied to the string, analyze the feasibility of inserting job n in the beginning of the string found on line [3.4.20]. The first test on line [3.4.22] is called the "Small Job Insertion Rule". This

rule stipulates that if HOURS is less than four (small job), then the value of WT must meet or exceed the value of HOURS. The next line [3.4.23], is called the "Big Job Insertion Rule". This rule stipulates that if HOURS is four hours or longer, then the value of WT must be at least four hours. These two rules combined equate to scheduling rule used by RSD decision makers that do not allow a big job ($\geq$ 4 hours) to be scheduled unless at least four hours of range time is available. If the job is small, then there needs to be at least as many hours of range time available as the job takes to complete. These rules are in place to cut down on the amount of travel time wasted traveling to and from work sites. If these rules return a TRUE, then the global variables START and END are assigned the values of BG and FS, respectively, regardless of the length of the job. Therefore, at this point the job can be inserted anywhere within that string, depending on the availability of manpower and equipment, which are tested in the next two tests. The last two rules on lines [3.4.24] and [3.4.25], called "Utility Rules". They eliminate jobs from consideration that have less range time than the job takes to complete, for small jobs, or that do not have at least four hours of range time for big jobs.

If either the small job or the big job insertion rules pass, the first and last value of the string is saved in the global variables START and END. The algorithm then continues onto the manpower availability test. If the Utility rules are TRUE, then the insertion subroutine is terminated and the top-level routine continues on to the next job.

3.4.4. Manpower Availability Test

The manpower availability test performs the same type of search as the range test but does it within the resource matrix M(t) (see line [3.4.27]). Local variables BG and FS are used as the start and finish of the manpower string. By applying the scheduling rules

on lines [3.4.28] through [3.4.32], BG and FS are compared to the global variables START and END; the start and end times of the maximum range string. The operation on line [3.4.27] looks for a value in the M(t) matrix, in the row "Type" and within columns (START - 1) to (END + 1) that is greater than or equal to the value WPH_n (Workers Per Hour) for job n.

Manpower Test
START
BG = FS = 25 [3.4.26]
FOR d_n search matrix M(t)["$TYPE_n$", i] for i = (START – 1) to (END + 1) [3.4.27]
for maximum length string of consecutive periods where M(t) ≥ "WPH_n"
SET BG = First period in string that M(t) ≥ "WPH_n"
SET FS = Last period in string that M(t) ≥ "WPH_n"

Figure 3.4 is a sample M(t) matrix. If for example, a job requires two Carpenter Workers Per Hour (WPH = 2), the algorithm could form a string seven periods long. Instead of searching the whole time frame provided by the matrix WW, this algorithm searches only those periods that have already been proven valid via the range test, namely the values START and END. Because the manpower must be scheduled during the time they are in transit to and from the work site (1 hour each way), the algorithm will search one hour before and one hour after the times found in the range test. In this example, by searching only between the periods (START – 1) and (END + 1) which are between "8" and "15", respectively, the maximum feasible string length for manpower is six periods with BG = "8" and FS = "13".

	Period									
TYPE	7	8	9	10	11	12	13	14	15	…
B										
C	4	4	4	4	2	2	2	0	0	0
E										
H										
S										
W										

Figure 3.4 Sample M(t) Matrix

After the algorithm finds the maximum string length of manpower resources for job d_n, the next step is to compare and shape the two strings to create a maximum combined feasible region. Logical equations [3.4.28] through [3.4.32] below create the maximum region so that the length of the manpower string overlaps the range resource string by one hour on each end (allows for worker transportation to and from the range). Equation [3.4.28] returns NO SOLUTION if no common manpower string was found within the bounds of the available range time. Line [3.4.26] sets the values BG and FS to 25 so that this logical check can be made. Equation [3.4.29], called the Containment Rule (CONR) returns a value of TRUE if the manpower resource overlaps the range resource by exactly one hour on each end (see Figure 3.5).

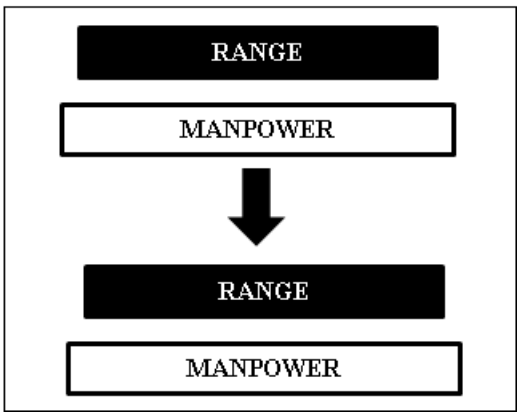

Figure 3.5 CONR

IF BG = FS = 25 THEN [3.4.28]
  RETURN NO SOLUTION
ELSEIF [START > BG] AND [END < FS] THEN [3.4.29]
  GOTO NEXT
ELSEIF [START ≤ BG] AND [END ≥ FS] THEN [3.4.30]
  (START = BG + 1) AND (END = FS - 1)
  GOTO NEXT
ELSEIF [START ≤ BG] AND [END < FS] THEN [3.4.31]
  START = BG + 1
ELSEIF [START > BG] AND [END ≥ FS] THEN [3.4.32]
  END = FS - 1
ENDIF

Equation [3.4.30], called the Both Overlap Rule (BOLR), determines if the range string overlaps the manpower string at both the lower and upper bounds. If the rule is TRUE, both START and END values are adjusted to a feasible range (See Figure 3.6).

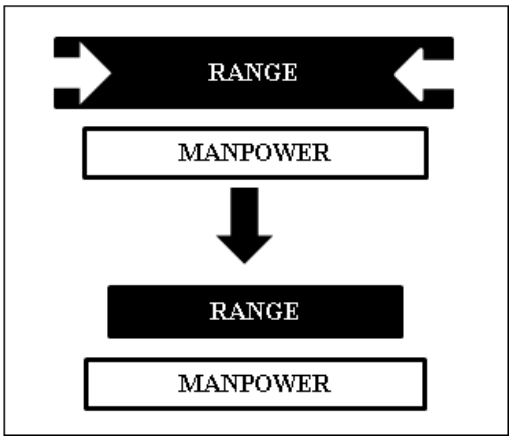

Figure 3.6 BOLR

Equations [3.4.31] and [3.4.32] are called the Left Overlap Rule (LOLR) and Right Overlap Rule (ROLR) respectively. They return a value of TRUE if the range string overlaps the manpower string on either the left or the right sides and adjusts the range string accordingly (see Figures 3.7 and 3.8)

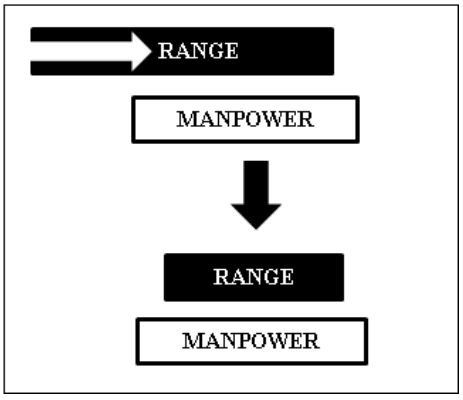

Figure 3.7 LOLR

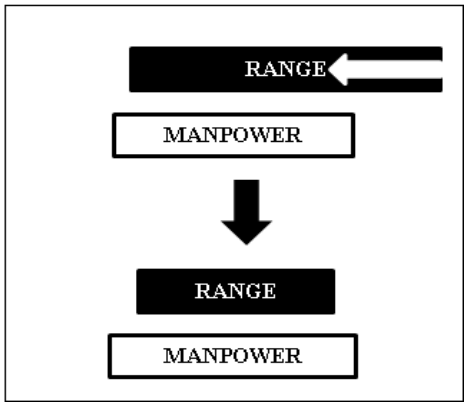

Figure 3.8 ROLR

The remaining operations of the manpower test determine if the newly adjusted range string passes the same Small Job Insertion and Big Job Insertion rules from the range test section (equations [3.4.22] and [3.4.23]). It is important to note that WT (equation [3.4.33]) is calculated using the values START and END instead of BG and FS, which is the span of the feasible range time.

WT = END – START + 1 [3.4.33]
IF [Small Job Insertion Rule] is TRUE THEN [3.4.34]
(START = START) AND (END = START + HOURS)
GOTO Equipment Test
IF [Big Job Insertion Rule] is TRUE THEN [3.4.35]
(START = START) AND (END = START + Min(WT, HOURS))
GOTO Equipment Test
IF [Utility Rules] are TRUE THEN [3.4.36]
RETURN NO SOLUTION
STOP

At this point the START and END times for the job j_n will be set if equations [3.4.34] or [3.4.35] have a value of TRUE. Otherwise, NO SOLUTION will be returned and the Insertion subroutine will terminate. If equation [3.4.34] (Small Job) is TRUE, then the job will start at the earliest possible time (START) and end when the job is done (START + HOURS). If equation [3.4.35] (Big Job) is TRUE, then the job will again

start as early as possible (START) and end at the expiration of either the available range time or after the job is complete [min(WT, HOURS)], whichever occurs first. After START and END times are set, the algorithm progresses to the equipment availability test.

3.4.5. Equipment Availability Test

The last check to make on available resources is the availability of heavy equipment. The vast majority of the heavy equipment is used exclusively by the heavy equipment operators (TYPE = H). Therefore, the "EQP" element of the array d_n will usually contain the empty set (∅) unless the job is tasked to the heavy equipment operators. If $(EQP)_n$ equals ∅, then no equipment on the equipment list (see Appendix C. List of Resources) is required and the job is ready to be scheduled (see equation [3.4.37]). If $(EQP)_n$ contains a list of required equipment ($EQP \neq \emptyset$) and every element of the array $(EQP)_n$ is also an element of the array E(t), then the job is ready to be scheduled as well (see equation [3.4.38]).

Equipment Test
START
  IF $(EQP)_n = \emptyset$ THEN [3.4.37]
    GOTO NEXT
  ELSEIF $(EQP)_n \subseteq E(t)$ THEN [3.4.38]
    GOTO NEXT
  ELSE [3.4.39]
    RETURN NO SOLUTION
  ENDIF

Otherwise, the array $(EQP)_n$ contains elements that are not also elements of the array E(t), so the job is not ready to be scheduled and the insertion subroutine will be terminated for job d_n (see equation [3.4.39]). The algorithm assumes that equipment is used only once per day. Heavy equipment is not particularly restrictive because there are

multiple pieces of redundant equipment. Additionally, due to the set up times and longer transportation times of the heavy equipment, scheduling equipment in a similar fashion to manpower would not be efficient.

NEXT
s_n = (SWN, SITE, TYPE, WPH, (EQP), START, END), $s_n \leftarrow d_n$ [3.4.40]
$d_n \leftarrow$ ("MH_n" | $MH_n = MH_n - Min[6, (END-START+1)]$) [3.4.41]
$M(t) \leftarrow (M(t) - WPH_n \mid Row = \text{"}TYPE_n\text{"}, Column = (START - 1) \text{ to } (END + 1))$
$R(t) \leftarrow (R(t) - 1 \mid Row = \text{"}SITE_n\text{"}, Column = (START \text{ to } END))$
$E(t) \leftarrow E(t) \setminus (EQP)_n$
STOP

Once a job d_n has passed all three tests (range, manpower, and equipment), it is ready to be scheduled. This algorithm schedules a job by mapping elements of d_n to s_n as well as adding the START and END times to s_n that were calculated in the manpower test (equations [3.4.34] and [3.4.35]). The START and END times found in those equations are the start and end of the range time for $SITE_n$ of job s_n. Equation [3.4.40] creates the array s_n, a member of the set S(t). After s_n is created, the last operation to accomplish is to adjust the level of remaining man-hours in d_n and the levels of available resources in the sets M(t), R(t), and E(t) (see four equations at line [3.4.41]).

Once SPSS algorithm steps through all jobs on the fifth day (Friday) it runs the Metric subroutine. This routine is explained in the next section.

3.5. Metrics Subroutine

The purpose of this study is to create an analytical scheduling tool that increases the number of jobs scheduled per week and decreases the amount of time it takes to generate an initial shell schedule via the current scheduling process. The following metrics will quantify any improvement in these objectives. There are two types of suggested metrics, objective and improvement metrics. The objective metrics directly

measure the objectives of this study and are meant to be used to compare the schedule constructed by the current scheduling methodology to the schedule constructed by the SPSS heuristic of Chapter 3. The improvement metrics are meant to be used as acceptance/rejection criteria for some type of interchange metaheuristic, such as the tabu or simulated annealing neighborhood search algorithms that were discussed in Chapter 2. This pseudo code for this subroutine is not explicitly stated, rather a discussion of each metric is provided below.

3.5.1. Objective Metrics

The first objective metric measures the daily percentage of active jobs scheduled by either method (current or SPSS algorithm). The cardinality of each days scheduled set (S) and active set (A) are used to calculate a percentage. Percentages for each method can be compared to determine if the SPSS algorithm schedules more jobs than the current method. This metric could also be used as an improvement metric.

Daily Percentage of Scheduled Jobs: $DPSJ_t = \frac{|S(t)|}{|A(t)|}$ [3.5.1]

The second objective metric measures the amount of time saved by using the algorithm to construct a schedule versus the current method.

Schedule Build Differential: SBD = (hours for current process) - (hours for algorithmic process) [3.5.2]

3.5.2. Improvement Metrics

The first improvement metric measures the Lateness (L) of a schedule, which is the cumulative number of days that the active jobs of the RSD are beyond their due date (late) for the entire week. The algorithm calculates the number of days that an active job

is beyond its scheduled due date by subtracting the current day ("Friday") from each jobs Dead Line (DL) only if the job has already exceeded its DL. It sums this value for every such job in the sets P and D on Friday.

$$\text{Lateness: } L = \sum_{\forall j} \left(DL_j - t \,\middle|\, t > DL_j,\ t=\text{"Friday"},\ p_j \in P(t) \text{ or } d_j \in D(t) \right) \qquad [3.5.3]$$

The last improvement metric calculates the number of unutilized man-hours for a schedule in a week. This metric is calculated by summing all the elements of each matrix M(t) for the entire week. It only calculates the unutilized man-hours for the time they are actually scheduled to work (i.e. within the limits of the matrix WW).

Unscheduled Man-hours:

$$UMH_{TYPE} = \sum_{\forall t} \sum_{i=SW}^{EW} M(t)\,(TYPE, i),\ \forall\ TYPE \in \{B,C,E,H,S,W\} \qquad [3.5.4]$$

3.6. Summary

The algorithm presented in this chapter contains all the features of a greedy method heuristic. The schedule that is created via its single-pass serial schedule generating scheme is guaranteed to produce at least a feasible solution. Metrics provided at the end of the chapter should determine if that schedule is a better solution than one generated via current methods. The next chapter uses a sample set to further investigate how the algorithm processes elements of an active set to construct a feasible schedule.

Chapter 4. Results and Analysis

4.1 Problem Sets

In this chapter, a six job example is used to help illustrate the processes within the algorithm and to show that this algorithm will produce a feasible schedule. This first section defines the problem sets used in the example. The example problem is outlined in Section 4.2 and in Section 4.3 an analysis of the algorithm's ability to construct a feasible schedule is discussed.

Figure 4.1 depicts the abbreviated form of the arrays a_j, p_j, d_j, and s_j that are used in the example problem.

$$\left.\begin{matrix} a_j \\ p_j \\ d_j \end{matrix}\right\} = \left[\mathrm{SWN}_j, \mathrm{MH}_j, \mathrm{PRED}_j, \mathrm{SUC}_j\right]$$

$$\left.s_j\right\} = \left[\mathrm{SWN}_j, \ \mathrm{START}_j, \mathrm{END}_j\right]$$

Figure 4.1 Example Arrays

Figure 4.2 shows the six sample arrays, a_1 through a_6, of the sample set A(1). For example, array a_1 has a Service/Work Order Number (SWN) of "E1", required man-hours (MH) = "6", no predecessor (PRED), and a successor with SWN = "C1", which is array a_2.

j	A(1)
1	[E1, 6, 0, C1]
2	[C1, 4, E1, H1]
3	[H1, 10, C1, 0]
4	[B1, 3, 0, 0]
5	[S1, 20, 0, H2]
6	[H2, 2, S1, 0]

Figure 4.2 Sample Arrays

4.2 Example Problem

This example contains six jobs that can be scheduled on a single range within the time periods 0800 to 1600. Figure 4.3 graphically shows how the sample six-array set of A(1) is processed by the prioritize subroutine which is run at the beginning of the SPSS routine. The left side of Figure 4.3 shows the sample arrays as they would appear in the active set A(1) once they are been created. The same arrays appear in their sorted order, in the far right column of Figure 4.3, in the prioritized set P(1) as they would once the prioritize subroutine is run. The arrays are prioritized in accordance with the dispatching rules discussed in Section 3.4.1. Of note, the precedence constraints are maintained after the jobs are prioritized and mapped to the set P(1).

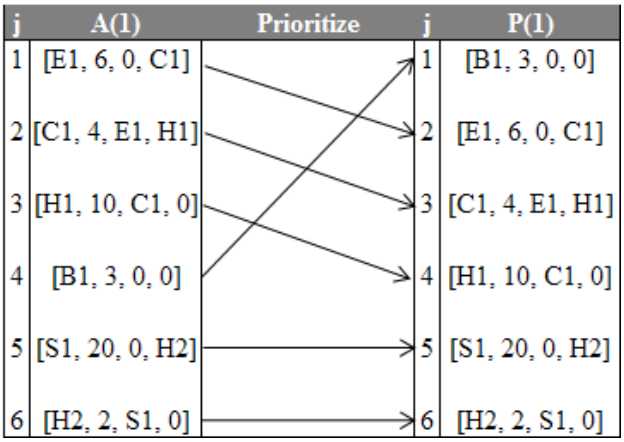

Figure 4.3 Prioritize Subroutine

Figures 4.4 through 4.7 graphically display the remaining operations of the SPSS routine. In Figure 4.4, the three columns P(1), D(1), and S(1) depict the status of the those sets for the first scheduled day after the second pass through the inner WHILE loop of the SPSS routine.

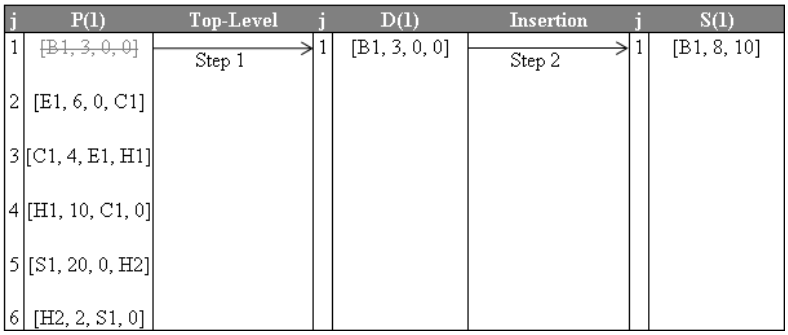

Figure 4.4 SPSS Routine (t = 1 "Monday", n = 1, 2)

The operation labeled "Step 1" corresponds to line [3.4.9] on the first pass through the inner WHILE loop (n = 1). This operation maps p_n to d_n if the job associated with p_n has no predecessors or if its predecessors have been completely scheduled. The former is true in this case. Additional operations on this first pass will remove array p_1 from the set P(1). Step 2 depicts the operation that takes place on the second pass (n still = 1) through the inner WHILE loop, of the SPSS routine, when the "IF" logic statement on line [3.4.7] returns a "TRUE". This is because array d_1 is a member of set D(1). Therefore, the insertion routine is performed on d_1. It is subsequently inserted into the set S(1). Step 3 in Figure 3.4 shows the remaining operations that are contained in the Insertion routine. Job B1's "MH_1" value is updated to reflect that it was completely scheduled during hours 8, 9, and 10 which correspond to clock time 0800:00 to 1059:59.

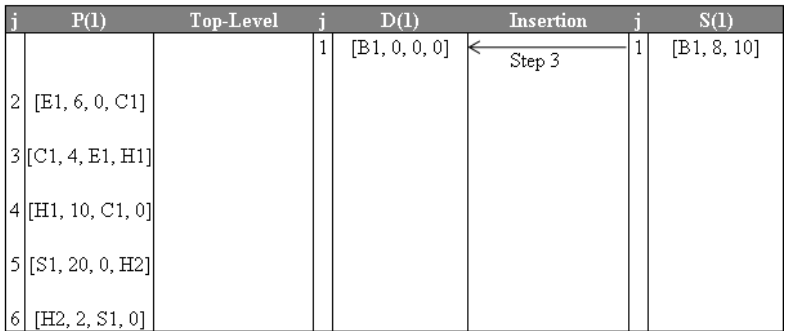

Figure 4.5 SPSS Routine (t = 1 "Monday", n = 2)

j	P(1)	Top-Level	j	D(1)	Insertion	j	S(1)
			1	[B1, 0, 0, 0]		1	[B1, 8, 10]
			2	[E1, 0, 0, C1]		2	[E1, 11, 13]
			3	[C1, 4, E1, H1]			
4	[H1, 10, C1, 0]						
			5	[S1, 16, 0, H2]		5	[S1, 14, 15]
6	[H2, 2, S1, 0]						

Figure 4.6 SPSS Routine (t = 1 “Monday”, n = 7)

Figure 4.6 depicts the results of the remaining iterations of the inner WHILE loop within the SPSS routine (n = 7) for day 1. At this point, all jobs have been processed for the first day. The results show that jobs B1 and E1 have been completely scheduled which allows job C1, which is a successor of E1, to be mapped into the decision set D(1). Sufficient resources did not exist to schedule any resource to job C1. Therefore, job H1 could not be pulled into the decision set D(1) because it is a successor of job C1. Job S1 was mapped into the decision set D(1) but was only partially scheduled as indicated by 16 man-hours remaining to completion. In this case, two hours of time is scheduled because two Surveyors are used per time period, for a total of four man-hours scheduled. Because job S1 is not completely scheduled job H2, a successor of S1, is not mapped to the decision set D(1). Since n = 7, which is greater than J (J = 6), the inner WHILE loop will STOP processing jobs for this first day (t = 1). The algorithm then continues on to line [3.4.11] where n is reset for the next day’s schedule and the day is incremented to Tuesday (t = 2). The resulting schedule for this example is job B1 from 0800 to 1059:59, job E1 from 1100 to 1359:59, and job S1 from 1400 to 1559:59.

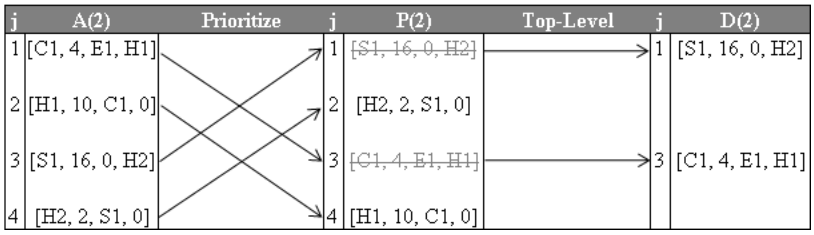

Figure 4.7 Prioritize Subroutine (t = 2 “Tuesday”, n = 0)

Figure 4.7 depicts the final operations that take place before the algorithm returns to the start of the outer WHILE loop to begin scheduling day 2. The first of these operations, on line [3.4.12], populate the a new active set A(2) with jobs from the previous priority set P(1) and any jobs in the previous decision set D(1) with man-hours greater than zero. The A(2) column of Figure 4.7 shows the results of that operation. Four jobs remain to be scheduled on the second day. The next line in the algorithm, [3.4.13], processes the arrays of A(2) in accordance with the Prioritize subroutine and places them in the new priority set P(2). Of note, the second and third priority jobs from the first day (Monday) are now the lowest priority jobs of the four remaining. This is done to illustrate how the variables Minimum Slack (MS) and Total Processing Time (TPT) can affect the priority ranking of jobs from day to day. This is an important aspect of this algorithm.

Lines [3.4.14] and [3.4.15] seed the new decision set D(2) with arrays that were previously in D(1) and then clears out those arrays from P(2). This is depicted in the columns P(2) and D(2) of Figure 4.7. The new decision set D(2) is seeded so that no further logic is necessary to use the same algorithm from day to day. In this case, job C1 would not have been pulled into the new decision set D(2) on day two because although the its predecessor has been completely scheduled, it is not a member of the set D(2) as the logic on line [3.4.9] in the inner WHILE loop stipulates.

4.3 Analysis

The example in Section 4.2 indicates that the greedy method employed in this research via the SPSS scheduling scheme will construct a valid schedule. As discussed in Chapter 2, this scheduling algorithm will produce an active schedule. In an active schedule, events cannot be scheduled any earlier without violating technological constraints, which is referred to as a left-shifted schedule. This is the reason each job is scheduled as early as possible within the feasible period of time (i.e. on the left side of the string). Because the algorithm inserts the highest priority job in the next available position, the algorithm will not utilize all the available range time and manpower. An improvement metaheuristic is required to "fill out" the empty portions of this schedule. Options for an improvement algorithm should be studied to maximize the number of jobs scheduled via this algorithm.

Chapter 5. Conclusions and Recommendations

This chapter summarizes the research of this graduate research project. The key points of the research are reemphasized and recommendations for future work are suggested.

5.1 Conclusions

This research explores possible methods of constructing a schedule for a complex resource constrained project scheduling problem. Multiple analytical solution methods are studied. A tractable solution method which solves to optimality is not feasible for this problem due to its computational complexity. Ultimately, a greedy heuristic method is used to construct a schedule for Eglin's Range Services Division. The algorithm created in this research utilizes a single-pass serial scheduling scheme along with a priority list to construct a feasible schedule. The priority lists are created with dispatching rules that , in prior studies, have proven to be both efficient and effective. Analysis of a sample problem in Chapter 4 indicates that this algorithm will construct an active and feasible schedule. Metrics to use for analytical comparisons between this schedule and one created by the current methodology are provided in Chapter 3. Furthermore, additional metrics are suggested that can be used as acceptance, rejection, or termination criteria for an improvement metaheuristic.

5.2 Recommendations for Future Work

This research was intended to provide a practical scheduling tool for the 46th Test Wing at Eglin AFB, FL. An algorithm was created that can be used to demonstrate an analytical solution method for a scheduling problem that is otherwise done by hand. The algorithm of this research can be coded into Microsoft Excel, with a minor amount of

Visual Basic for Applications (VBA) coding. Once this algorithm is coded, the following future work is recommended:

- Incorporate a metaheuristic improvement algorithm to the schedule construction algorithm of this research. There will be un-utilized range time and manpower left over after the SPSS routine is run. An Interchange metaheuristic could possibly create additional space for the scheduling of more jobs.
- The algorithm needs to be verified and validated against the current scheduling process to ensure it adequately models the current scheduling process.
- Once the algorithm is verified and validated, it should be tested to determine to what degree it improves upon the current scheduling process, with respect to the number of jobs it can schedule and the time it takes to create an initial shell.
- Ultimately, a verified and validated scheduling tool should be incorporated into the current 46th Test Wing's current scheduling suite called the Center Scheduling Enterprise (CSE).

Appendix A. List of Acronyms

AAC	Air Armament Center
AFB	Air Force Base
AFMC	Air Force Materiel Command
BM	Building Maintenance
CSE	Center Scheduling Enterprise
DL	Dead Line
DoD	Department of Defense
DMS	Direct Mission Support
ETTC	Eglin Test and Training Center
EQP	Equipment
EW	End Work
IWO	Internal Work Order
MH	Man-Hours
MS	Minimum Slack
PRED	Priority Number
PRI	Predecessor
RC	Range Construction
RCPSP	Resource Constrained Project Scheduling Problem
RSD	Range Support Division
SUC	Successor
SPSS	Single-Pass Serial Schedule

SW	Start Work
SWN	Service / Work Order Number
TE	Test Engineer
TO	Task Order
TW	Test Wing
USAF	United States Air Force
WPH	Workers Per Hour
WT	Work Time
WW	Work Week

Appendix B. InDyne Maximum Work Time Policy

The goal of the InDyne ETTC O&M Contract Maximum Work Time Policy is to establish guidelines for supervisors to consider when making decisions about overtime. Supervisors should look ahead, plan man hours against workload and limit the use of overtime as the shock absorber for peaks and valleys in workload. Although all of these restrictions can be waived by the indicated level of authority, supervisors should manage their people within these restrictions as much as possible.

- The Maximum Work Time per day shall be 12 hours; the Director/Office Manager can pre-approve up to 16 hours per day.

- The Maximum Work Time per week shall be 60 hours; the Director/Office Manager can pre-approve up to 72 hours per week.

- The General Manager's approval is required before an employee can work periods exceeding those the Directors/Office Managers can approve.

It is also important to ensure there is adequate rest period between shifts. Supervisors should manage work schedules so employees have, at a minimum, time enough to drive home, get some rest and drive back to work before they start the next shift.

- The normal time between shifts shall be at least 8 hours; the Director/Office Manager can pre-approve periods less than 8 hours.

Appendix C. List of Resources

Resource			
Manpower		**Current #**	**Helps**
	Electricians - E	8	W, C, S, H, B
	Welders - W	4	E, C, S, H, B
	Carpenters - C	6	E, W, S, H, B
	Surveyors - S	4	E, W, C, H, B
	Hvy Equip Operators- H	7	E, W, C, S, B
	Building Maintenance - B	5	E, W, C, S, H
Equipment		**Current #**	**Requires**
	Electricians		
	Trucks	4	
	Bucket Truck	1	
	Bucket Truck w/pole set	1	
	Welders		
	Trucks	2	
	Welders	4	
	Portable Welders	5	
	Carpenters		
	Trucks	2	
	1.5-ton Flatbed	1	
	Utility Trailer	1	
	Surveyors		
	Trucks	2	
	Hvy Equip		
	Trucks	2	
	1.5-ton Trucks	2	
	Tractor	4	
	Trailers	4	Tractor
	60-ton Lowboy Trailer	3	Tractor
	Drop-neck Trailer	1	Tractor
	30-ton 40' Trailer	2	Tractor
	25-ton Crane (New)	1	Tractor
			Drop-neck Trailer
	25-ton Crane (Old)	1	
	100-ton Crane	1	
	Loaders	3	60-ton Trailer or
			Drop-neck Trailer
	Forklifts	4	60-ton Trailer
			Drop-neck Trailer
	Bulldozer	1	Dump Truck w/20-ton Trailer
	Dump Truck w/20-ton Trailer	1	
	Road Grader	1	
	Water Truck	1	
	Wrecker	1	
	Farm Tractor	1	Any Trailer

Appendix D. Single-Pass Serial Scheduling Algorithm

Define Parameters and Variables:

Parameters:

t ≡ Current scheduling day (MM:DD:YYYY), t ∈ {(1:1:2009), (1:2:2009),...} [3.2.1]

WW ≡ Work Week, 2x5 matrix of starting (SW) and ending (EW) times for schedulable work during day t for the weekly schedule [3.2.2]

Variables:

$K_r(t)$ ≡ Array of available resources for each day t, r ∈ {R, M, E} [3.2.3]
- R(t) ≡ Matrix of range availability per period for day t
- M(t) ≡ Matrix of available man-hours available per period for day t
- E(t) ≡ Array of heavy equipment available for day (t)

A(t) ≡ Active Set of arrays a_j for each job j in a scheduling day t, j ∈ {1, 2, 3,…, J} [3.2.4]
- a_j ≡ (SWN, SITE, PRI, MH, WPH, TYPE, (EQP), PRED, SUC, DL, MS)
- SWN_j ≡ Service Request or Work Order Number of job j
- $SITE_j$ ≡ Work site identifier
- PRI_j ≡ Priority number, ∈ {1, 2, 3}
- MH_j ≡ Man Hours required for job j, ∈ {0, 1, 2, 3,...}
- WPH_j ≡ Workers Per Hour required for job j, ∈ {1, 2, 3,...}
- $TYPE_j$ ≡ Type of worker required for job j, ∈ {B, C, E, H, S, W}
  - B = Building Maintenance; C = Carpenter; E = Electrician;
  - H = Heavy Equipment Operator; S = Surveyor; W = Welder
- $(EQP)_j$ ≡ Array of Heavy Equipment required for job j
- $PRED_j$ ≡ Job that must be completed prior the start of job j (Predecessor)
- SUC_j ≡ Job that can start only once job j is completed (Successor)
- DL_j ≡ Dead Line of job j, ∈ {(MM:DD:YYYY)}
- MS_j ≡ Minimum Slack of job j; $MS_j \equiv DL_j - (t + TPT_j)$ [3.2.5]
  - TPT_j ≡ Total Processing Time = $MH_j + \sum$MH of all Successors of job j

P(t) ≡ Prioritized, set of arrays p_j of sorted elements of A(t); $p_j \leftarrow a_j, \forall j$ [3.2.6]

D(t) ≡ Decision, set of arrays d_j of schedulable jobs from P(t); $d_j \leftarrow p_j$, [3.2.7]

S(t) ≡ Scheduled, set of arrays s_j of scheduled jobs [3.2.8]
- s_j ≡ (SWN, SITE, TYPE, WPH, (EQP), START, END)
- START = Start of work for job j on $SITE_j$; END = End of work for job j on $SITE_j$

$WT_j = MH_j/WPH_j$; Work Time, hours required at $SITE_j$ to complete job j, ∈ ℝ [3.2.9]

Appendix D. Single-Pass Serial Scheduling Algorithm Cont.

Single Pass Serial Scheduling Routine:

START
- $n = 0$ [3.4.1]
- $t = t_o$ [3.4.2]
- $A(t) = P(t) = D(t) = S(t) = \emptyset$ [3.4.3]
- RUN Prioritize Subroutine [3.4.4]
- WHILE $t \leq (t_o + 5 \text{ days})$ DO [3.4.5]
- BEGIN
  - WHILE $n \leq J$ DO [3.4.6]
  - BEGIN
    - IF $d_n \in D(t)$ THEN [3.4.7]
      - RUN Insertion Subroutine
    - ENDIF
    - $n = n + 1$ [3.4.8]
    - $D(t) \leftarrow \{p_n \mid$ "$PRED_n$" = 0 OR "$PRED_n$" = "SWN_i" with "MH_i" = 0$\}$; [3.4.9]
      $\forall$ i, i $\in \{1, 2, 3, \ldots, J\}$, $i \neq n$, $d_n \leftarrow p_n$
    - $P(t) \equiv \{p_j \mid$ "SWN_j" $\notin D(t)\}$, $\forall$ j [3.4.10]
  - END WHILE
- $n = 0$
- $t = t + (1 \text{ day})$ [3.4.11]
- $A(t) \leftarrow P(t\text{-}1) \cup \{d_j \in D(t\text{-}1) \mid$ "MH_j" $> 0\}$, $\forall$ j, $a_j \leftarrow p_j$, $a_j \leftarrow d_j$ [3.4.12]
- RUN Prioritize Subroutine [3.4.13]
- $D(t) \leftarrow \{p_j \in P(t) \mid$ "SWN_j" = "SWN_i" of $d_i \in D(t\text{-}1)\}$, $\forall$ i, j [3.4.14]
- $P(t) \equiv \{p_j \mid$ "SWN_j" $\notin D(t)\}$, $\forall$ j [3.4.15]
- END WHILE
- Metric Subroutine

STOP

Prioritize Subroutine:

If $t = t_o$ THEN
- Create J arrays of set A(t) from job and user input databases [3.4.16]
- Number the elements of a_j sequentially, j = 1, 2, 3, …

ENDIF

$J = |A(t)|$

Sort the J elements of A(t) by: [3.4.17]

4) Increasing order of priority number (PRI_j)*
   *A Successor will have same priority number as their predecessor
5) Then by increasing order of Minimum Slack (MS_j)
6) Then by decreasing order of Total Processing Time (TPT_j)

Place ordered elements of A(t) in P(t), sequentially renumber the J elements of P(t);

$p_j \leftarrow a_i$, i = 1, 2,..., J, j = 1, 2, 3, …, J [3.4.18]

Appendix D. Single-Pass Serial Scheduling Algorithm Cont.

Insertion Subroutine:

For job d_n: SET HOURS = $\left\lceil \frac{MH_n}{WPH_n} \right\rceil$ [3.4.19]

Range Test

START

FOR d_n Search R(t)(Row = "$SITE_n$") from (SW(t) + 1) to (EW(t) -1) [3.4.20]
to find the maximum length string of consecutive free periods

SET BG = First free period in string
SET FS = Last free period in string
WT = FS – BG + 1 [3.4.21]

IF (HOURS < 4) AND (WT ≥ HOURS) THEN *[Small Job Insertion Rule]* [3.4.22]
(START = BG) AND (END = FS)
GOTO Manpower Test

ELSEIF (HOURS ≥ 4) AND (WT ≥ 4) *[Big Job Insertion Rule]* [3.4.23]
(START = BG) AND (END = FS)
GOTO Manpower Test

ELSEIF (HOURS < 4) AND (WT < HOURS) *[Utility Rule]* [3.4.24]
RETURN NO SOLUTION

ELSEIF (HOURS ≥ 4) AND (WT < 4) *[Utility Rule]* [3.4.25]
RETURN NO SOLUTION

ENDIF

STOP

Appendix D. Single-Pass Serial Scheduling Algorithm Cont.

Manpower Test

START

BG = FS = 25 [3.4.26]

FOR d_n search matrix M(t)["$TYPE_n$", i] for i = (START – 1) to (END + 1) [3.4.27]
for maximum length string of consecutive periods where M(t) ≥ "WPH_n"
SET BG = First period in string that M(t) ≥ "WPH_n"
SET FS = Last period in string that M(t) ≥ "WPH_n"

IF BG = FS = 25 THEN *[Infeasible]* [3.4.28]
RETURN NO SOLUTION

ELSEIF [START > BG] AND [END < FS] THEN *[Both Overlap Rule]* [3.4.29]
GOTO NEXT

ELSEIF [START ≤ BG] AND [END ≥ FS] THEN *[Contain Rule]* [3.4.30]
(START = BG + 1) AND (END = FS - 1)
GOTO NEXT

ELSEIF [START ≤ BG] AND [END < FS] THEN *[Left Overlap Rule]* [3.4.31]
START = BG + 1

ELSEIF [START > BG] AND [END ≥ FS] THEN *[Right Overlap Rule]* [3.4.32]
END = FS - 1

ENDIF

WT = END – START + 1 [3.4.33]

IF [Small Job Insertion Rule] is TRUE THEN [3.4.34]
(START = START) AND (END = START + HOURS)
GOTO Equipment Test

IF [Big Job Insertion Rule] is TRUE THEN [3.4.36]
(START = START) AND (END = START + Min(WT, HOURS))
GOTO Equipment Test

IF [Utility Rules] are TRUE THEN
RETURN NO SOLUTION

STOP

Appendix D. Single-Pass Serial Scheduling Algorithm Cont.

Equipment Test

START

IF $(EQP)_n = \emptyset$ THEN [3.4.37]

GOTO NEXT

ELSEIF $(EQP)_n \subseteq E(t)$ THEN [3.4.38]

GOTO NEXT

ELSE [3.4.39]

RETURN NO SOLUTION

ENDIF

NEXT

$s_n = (SWN, SITE, TYPE, WPH, (EQP), START, END), s_n \leftarrow d_n$ [3.4.40]

$d_n \leftarrow (\text{“}MH_n\text{”} \mid MH_n = MH_n - Min[6, (END\text{-}START+1)])$ [3.4.41]

$M(t) \leftarrow (M(t) - WPH_n \mid Row = \text{“}TYPE_n\text{”}, Column = (START - 1) \text{ to } (END + 1))$

$R(t) \leftarrow (R(t) - 1 \mid Row = \text{“}SITE_n\text{”}, Column = (START \text{ to } END))$

$E(t) \leftarrow E(t) \setminus (EQP)_n$

STOP

Appendix D. Single-Pass Serial Scheduling Algorithm Cont.

Metric Subroutine:

Calculate the following metrics after the SPSS routine cycles through the last day (t = Friday):

Objective Metrics

Daily Percentage of Scheduled Jobs: $DPSJ_t = \frac{|S(t)|}{|A(t)|}$ [3.5.1]

Schedule Build Differential: SBD = (hours for current process) - (hours for algorithmic process) [3.5.2]

Improvement Metrics

Lateness: $L = \sum_{\forall j} \left(DL_j - t \,\middle|\, t > DL_j, \text{t="Friday"}, p_j \in P(t) \text{ or } d_j \in D(t) \right)$ [3.5.3]

Unscheduled Man-hours:

$$UMH_{TYPE} = \sum_{\forall t} \sum_{i=SW}^{EW} M(t)\,(TYPE, i), \ \forall\ TYPE \in \{B,C,E,H,S,W\} \quad [3.5.4]$$

Appendix E. Blue Dart

Optimizing Your Schedule

In a perfect world, organizations would always have enough time and resources to accomplish every task they are assigned. In reality, most organizations are overwhelmed by the number of tasks they are appointed as well as constrained with too few resources to accomplish them. Our country's current economic situation dictates that everyone do more with less; unfortunately, the Air Force is no stranger to this issue. Effectively scheduling our dwindling resources is the linchpin to meeting mission requirements given our current situation.

Typically, Air Force organizations are constrained in the amount of work they can accomplish by: man-hours, equipment, material, workspace, and time. For example, a flying Squadron would never schedule more flying sorties in a day than its pilots are capable of flying or its Maintenance Squadron is capable of producing. To ensure all the work gets done, given their constraints, most organizations employ labor intensive scheduling processes. Unfortunately, the scheduling approach most organizations employ frequently results in an ineffective schedule.

Organizations, particularly Air Force organizations, live and die by how well they schedule their limited resources to accomplish mission oriented tasks. A simplistic criterion of a successful schedule is whether all the required tasks are scheduled to be completed prior to their due dates. A schedule built with only this goal in mind is referred to as a "valid" schedule. On the other hand, an optimal schedule is designed to efficiently utilize available resources to produce the maximum amount of work possible. Although there may be reasons why organizations decide not to execute an optimal

schedule, but an optimal schedule does provide a best case from which an execution schedule can be built.

Air Force organizations need to adopt scheduling practices that will allow them to build optimal or even nearly optimal schedules rather than just valid schedules. It is very difficult for their senior decision maker to understand when a true capacity limit has been reached if an organization uses merely a valid schedule. Experience and intuition are commonly used tools to assess when limits have been reached, but as is often seen, experience and intuition are highly subjective and readily dismissed by decision makers outside the organization.

Optimally built schedules provide organizations with the information they require to effectively fight for resources when they are over tasked. More importantly, optimally built schedules provide organizations with the ability to precisely quantify how much work they can accomplish before such a limit, or upper bound, is reached. For example, if your bagger at the grocery store determines that he has packed as many groceries into a brown paper bag as possible [optimality], how could you justify asking him to pack one more thing without first acknowledging that something must be removed? Information based on an optimal schedule, which is a repeatable and mathematically defendable process, provides decision makers and organizations alike with the information they need to make more coherent decisions on task priorities when something must be "removed from the bag."

Within the field of Operations Research, there are several very useful and easily applied tools that allow organizations to build optimal schedules. One powerful tool often used in scheduling applications is linear programming. Linear programming is a

mathematical method that can produce optimal solutions to many types of scheduling problems, both big and small. These tools can be easily programmed into user friendly and readily available spreadsheet programs such as Microsoft Excel or incorporated into existing scheduling software packages such as Patriot Excalibur.

Organizations that want to maximize their capacity should build their own user-friendly and computer aided scheduling tools. Armed with these powerful tools, organizations will have at their disposal a repeatable method of realistically scheduling as much as possible without over-tasking their resources. Furthermore, these tools will provide decision makers with the information they need to make well informed operational decisions. After all, wouldn't it be nice to focus on accomplishing the mission instead of constantly chasing a schedule?

Contact your MAJCOM A-9 or the Air Force Institute of Technology Center for Operational Analysis (AFIT-COA) for more information on simple ways to improve your scheduling processes. Major Matt Liljenstolpe is an AFIT student in the Operations Analysis program.

Keywords: Scheduling, Linear Programming, Air Force Institute of Technology Center for Operational Analysis, A-9 Analysis and Lessons Learned

Bibliography

Artigues, C., Demassey, S., Neron, E. (eds.). 2008. **Resource-constrained project scheduling: Models, algorithms, extensions and applications.** Hoboken, NJ: Wiley.

Air Armament Center Fact Sheet. 96th Air Base Wing Public Affairs Office, Eglin AFB, FL. 7 March 2009 http://www.eglin.af.mil/library/factsheets/factsheet.asp?id=6476.

Burns, Dave. Supervisor of Range Support Division, InDyne Inc., Eglin AFB, FL. Personal Interview. 22 January 2009.

French, S. 1982. **Sequencing and scheduling: An introduction to the mathematics of the job-shop.** West Sussex, England: Ellis Horwood.

Heald, James. General Manager of InDyne Inc., Eglin AFB, FL. Personal Interview. 14 January 2009.

Kolisch, R. 1996. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. **European Journal of Operational Research,** 90: 320-333.

Murty, K.G. 1995. **Operations research: Deterministic optimization models.** Englewood Cliffs, N.J.: Prentice-Hall.

Pinedo, M. 2008. **Scheduling: Theory, algorithms, and systems 3rd ed.** New York, NY: Springer.

Yang, K.K. 1998. A comparison of dispatching rules for executing a resource-constrained project with estimated activity durations. **Omega International Journal of Management Science,** 26(6): 729-738.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
18-06-2009	Graduate Research Project	23-06-2008 - 18-06-2009

4. TITLE AND SUBTITLE
SINGLE-PASS SERIAL SCHEDULING HEURISTIC FOR EGLIN AFB RANGE SERVICES DIVISION SCHEDULE

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Liljenstolpe, Matthew, Major, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Street, Building 642
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT-IOA-ENS-09C-02

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFMC 46 TW/OA Attn: Mr. Greg Hutto
Room 229 Bldg 1
West D St
Eglin AFB, Florida 32542-5000 Phone: (850) 882-4646
e-mail: Gregory.Hutto@Eglin.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)
N/A

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
N/A

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The Air Armament Center (AAC) located at Eglin Air Force Base (AFB) Florida, conducts test and evaluation of United States Air Force (USAF) weapons systems. To enable this, the AAC operates the Eglin Test and Training Complex (ETTC), the largest test range in the United States. InDyne Corporation's Range Services Division (RSD) builds and maintains the infrastructure necessary to conduct world class test and training on the ETTC.

The purpose of this research is to create a scheduling tool for the RSD that maximizes the number of prioritized jobs scheduled and reduces the man-hours required to create a weekly schedule without exceeding a job's deadline, manpower, or equipment constraints. RSD's schedule belongs to a class of scheduling problems called Resource Constrained Project scheduling Problems (RCPSP). RCPSPs attempt to schedule activities of either a known (deterministic) or variable (stochastic) duration in a defined sequence given a finite amount of resources. Many analytical methods have been created to solve these types of scheduling problems. Analytical solution methods which guarantee optimal solutions were not feasible due to the computational complexity of this RCPSP. Instead, a greedy solution method is explored that uses a single-pass serial scheduling algorithm.

A schedule construction algorithm is provided in the form of pseudo code to enable further research and development of a scheduling tool for this RCPSP. Research on a schedule improvement metaheuristics and coding of the complete algorithm is required before it can be assimilated into existing scheduling software.

15. SUBJECT TERMS
Resource Constrained Project Scheduling Problem, Greedy Method, Precedence Constraints, Preemption, Non-Deterministic Polynomial, Polynomial, Computational Complexity, Dispatching Rules, Priority List, Single Pass Serial Scheduling Algorithm, Heuristics, Construction Algorithm, Metaheuristic Improvement Algorithm, Eglin Test and Training Complex, 46th Test Wing, Pseudo Code, Metrics.

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Prof J.T. Moore, ENS
U	U	U	UU	80	19b. TELEPHONE NUMBER *(Include area code)* DSN: 785-3636, ext. 4528, e-mail: James.Moore@AFIT.edu